CryptoGateway Documentation

Adrian Bedard Jonathan Bedard

April 18, 2016

Contents

Part I

CryptoGateway Library

Chapter 1

Introduction

The CryptoGateway library contains classes which handle cryptography. CryptoGateway is designed as an open source library, so much of the cryptography within the library is relatively simple. CryptoGateway is not meant to define cryptography to be used widely, rather, it is meant to provide a series of generalized hooks and interfaces which can be extended to various cryptographic algorithms.

1.1 Namespace

CryptoGateway uses the crypto namespace. The crypto namespace is designed for class, functions and constants related to cryptography. CrytpoGateway depends on many of the tools defined in the os namespace. Additionally, the crypto namespace contains a series of nested namespaces which help to disambiguate constants.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 binaryEncryption.cpp File Reference

Implementation of binary encryption files.

3.1.1 Detailed Description

Implementation of binary encryption files.

Author

Jonathan Bedard

Date

4/18/2016

Bug None

Implements the binary encryption files. Consult **binaryEncryption.h** (p. 7) for details on using these classes.

3.2 binaryEncryption.h File Reference

Definition of binary encryption files.

Classes

- class **crypto::binaryEncryptor**
  Encrypted binary file output.
- class **crypto::binaryDecryptor**
  Encrypted binary file output.

Namespaces

- **crypto**

3.2.1 Detailed Description

Definition of binary encryption files.

Author

Jonathan Bedard

Date

3/7/2016

Bug None

Provides an interface to dump and retrieve data from an encrypted binary file without concern as to the encryption algorithm used.

3.3 c_BaseTen.c File Reference

Implementation of base-10 algorithms.

3.3.1 Detailed Description

Implementation of base-10 algorithms.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements all of the basic functionality of a base-10 integer. All integer operations, both basic and otherwise, are implemented in this file.

3.4 c_BaseTen.h File Reference

Base-10 number functions.

Functions

- struct **numberType** ∗ **buildBaseTenType** ()

  Construct a base-10 number.
- int **base10Addition** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)

  Base-10 addition.
- int **base10Subtraction** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)

Base-10 subtraction.

- int **base10Multiplication** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 multiplication.
- int **base10Division** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 division.
- int **base10Modulo** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 modulo.
- int **base10Exponentiation** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 exponentiation.
- int **base10ModuloExponentiation** (const uint32_t *src1, const uint32_t *src2, const uint32_t *src3, uint32_t *dest, uint16_t length)
- int **base10GCD** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)
- int **base10ModInverse** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)
- int **primeTest** (const uint32_t *src1, uint16_t test_iteration, uint16_t length)

3.4.1 Detailed Description

Base-10 number functions.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains functions which define a base-10 integer. There functions are bound to a number type.

3.4.2 Function Documentation

int base10Addition (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 addition.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Division (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 division.

This function takes in two arrays which represent base-10 numbers, preforms src1/src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Exponentiation (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 exponentiation.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10GCD (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

int base10ModInverse (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

int base10Modulo (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 modulo.

This function takes in two arrays which represent base-10 numbers, preforms src1src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10ModuloExponentiation (const uint32_t * src1, const uint32_t * src2, const uint32_t * src3, uint32_t * dest, uint16_t length)

int base10Multiplication (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 multiplication.

This function takes in two arrays which represent base-10 numbers, preforms src1*src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Subtraction (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 subtraction.

This function takes in two arrays which represent base-10 numbers, preforms src1-src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

struct **numberType**∗ buildBaseTenType ( )

Construct a base-10 number.

This function will return a **numberType** (p. 191) pointer defining the function pointers for a base-10 number. Note that the resulting pointer points to a structure which is static to the **c_BaseTen.c** (p. 8) file.

Returns

Pointer to **numberType** (p. 191) of type base-10

int primeTest (const uint32_t ∗ src1, uint16_t test_iteration, uint16_t length)

3.5 c_cryptoTesting.cpp File Reference

Implementation for C file testing.

3.5.1 Detailed Description

Implementation for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements test suites which are testing raw C code. This file currently tests the Base-↩ Ten suite.

3.6 c_cryptoTesting.h File Reference

Header for C file testing.

3.6.1 Detailed Description

Header for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This header is meant for the test suites which are testing raw C code. This header currently contains the Base-Ten suite.

3.7 c_numberDefinitions.c File Reference

Implementation of basic number.

3.7.1 Detailed Description

Implementation of basic number.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Most numerical operations must be defined by the specific number type, but a select few are generally applicable across all number types, these are implemented here.

3.8 c_numberDefinitions.h File Reference

Basic number declarations.

Classes

- struct **numberType**

  Number type function structure.

Typedefs

- typedef int(∗ **operatorFunction**) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

  Operator function typedef.
- typedef int(∗ **tripleCalculation**) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

  Triple operator function typedef.
- typedef int(∗ **shiftFunction**) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)

  Shift operator function typedef.
- typedef int(∗ **compareFunction**) (const uint32_t ∗, const uint32_t ∗, uint16_t)

  Comparison function typedef.

Functions

- struct **numberType** ∗ **buildNullNumberType** ()

  Construct a NULL number.
- int **standardCompare** (const uint32_t ∗src1, const uint32_t ∗src2, uint16_t length)

  Standard comparision.
- int **standardRightShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)

  Right shift.
- int **standardLeftShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)

  Left shift.

3.8.1 Detailed Description

Basic number declarations.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains function typedefs used for various number operations and defines a few nearly universal numerical functions.

3.8.2 Typedef Documentation

typedef int(∗ compareFunction) (const uint32_t ∗, const uint32_t ∗, uint16_t)

Comparison function typedef.

This function typedef defines a function which takes in two arrays which represent numbers and then compares them.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint32_t*∗	Argument 2
in	*uint16_t*	size

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

typedef int(∗ operatorFunction) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Operator function typedef.

This function typedef defines a function which takes in two arrays which represent numbers, preform some operation on the pair and then output the result to a third array.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint32_t*∗	Argument 2
out	*uint32_t*∗	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

typedef int(∗ shiftFunction) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)

Shift operator function typedef.

This function typedef defines a function which takes in an array representing a number, shifts it the provided number of bits and outputs the result into the second array.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint16_t*	Bits to shift
out	*uint32_t*∗	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

typedef int(∗ tripleCalculation) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Triple operator function typedef.

This function typedef defines a function which takes in three arrays which represent numbers, preform some operation on the triple and then output the result to a fourth array.

Parameters

in	*uint32_t**	Argument 1
in	*uint32_t**	Argument 2
in	*uint32_t**	Argument 3
out	*uint32_t**	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

3.8.3 Function Documentation

struct **numberType*** buildNullNumberType ( )

Construct a NULL number.

This function will return a **numberType** (p. 191) pointer defining the function pointers for a NULL number. Note that the resulting pointer points to a structure which is static to the **c_number↩ Definitions.c** (p. 13) file.

Returns

Pointer to **numberType** (p. 191) of type NULL

int standardCompare (const uint32_t * src1, const uint32_t * src2, uint16_t length)

Standard comparision.

This function takes in two arrays which represent numbers and then compares them.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
in	*length*	Number of uint32_t in the arrays

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

int standardLeftShift (const uint32_t * src1, uint16_t src2, uint32_t * dest, uint16_t length)

Left shift.

Shifts the bits in src1 in the left direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Bits to shift
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int standardRightShift (const uint32_t * src1, uint16_t src2, uint32_t * dest, uint16_t length)

Right shift.

Shifts the bits in src1 in the right direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Bits to shift
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

3.9 cryptoCConstants.h File Reference

Extern declarations of C constants.

Variables

- const int **crypto_numbertype_default**

  Default number ID.
- const int **crypto_numbertype_base10**

  Base-10 number ID.
- const char * **crypto_numbername_default**

  Default number marker.
- const char * **crypto_numbername_base10**

  Base-10 number marker.

3.9.1 Detailed Description

Extern declarations of C constants.
Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Declares a number of constants needed by both the C numerical algorithms and by C++ number classes.

3.9.2 Variable Documentation

const char* crypto_numbername_base10

Base-10 number marker.
This constant is "Base 10 Type". It represents a number of type base-10, or standard integer.

const char* crypto_numbername_default

Default number marker.
This constant is "NULL Type". It represents an untyped number.

const int crypto_numbertype_base10

Base-10 number ID.
This constant is 1. It represents a number of type base-10, or standard integer.

const int crypto_numbertype_default

Default number ID.
This constant is 0. It represents an untyped number.

3.10 cryptoCHeaders.h File Reference

Collected headers for C source code.

3.10.1 Detailed Description

Collected headers for C source code.
Author

Jonathan Bedard

Date

2/20/2016

Bug None

3.11 cryptoConstants.cpp File Reference

Implementation of CryptoGateway constants.

3.11.1 Detailed Description

Implementation of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Binds all of the scoped constants used by CryptoGateway. The nested namespaces ensure that there is no ambiguity as to the purpose and nature of the constants.

3.12 cryptoConstants.h File Reference

Extern definitions of CryptoGateway constants.

3.12.1 Detailed Description

Extern definitions of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Consult **cryptoConstants.cpp** (p. 19) for details. This file merely defines extern references to the global constants in **cryptoConstants.cpp** (p. 19).

3.13 cryptoCSource.cpp File Reference

Implementation of all C code.

3.13.1 Detailed Description

Implementation of all C code.

Author

Jonathan Bedard

Date

2/13/2016

Bug No known bugs.

This file includes all of the .c files needed for this library. It allows the CMake scripts for this project to be entirely C++ while still includeing raw C code.

3.14 cryptoError.cpp File Reference

Implementation of error sender and listener.

3.14.1 Detailed Description

Implementation of error sender and listener.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Implements the error sender and listeners. These classes allow for managing the throwing of **crypto::errorPointer** (p. 50). Consult **cryptoError.h** (p. 20) for details.

3.15 cryptoError.h File Reference

Declaration of cryptographic errors.

Classes

- class **crypto::error**
  Sortable exception.
- class **crypto::passwordSmallError**
  Symmetric key too small.
- class **crypto::passwordLargeError**
  Symmetric key too big.

- class **crypto::bufferSmallError**

  Buffer too small.
- class **crypto::bufferLargeError**

  Buffer too large.
- class **crypto::insertionFailed**

  ADS Insertion Failed.
- class **crypto::customError**

  *Custom **crypto::error*** *(p. 79).*
- class **crypto::fileOpenError**

  File open error.
- class **crypto::fileFormatError**

  File format error.
- class **crypto::illegalAlgorithmBind**

  Algorithm bound failure.
- class **crypto::hashCompareError**

  Hash mis-match.
- class **crypto::hashGenerationError**

  Hash generation error.
- class **crypto::actionOnFileError**

  File error.
- class **crypto::actionOnFileClosed**

  File closed error.
- class **crypto::publicKeySizeWrong**

  Public-key size error.
- class **crypto::keyMissing**

  Key missing error.
- class **crypto::NULLPublicKey**

  NULL public-key error.
- class **crypto::NULLDataError**

  NULL data error.
- class **crypto::NULLMaster**

  NULL master error.
- class **crypto::masterMismatch**

  Master mis-match.
- class **crypto::unknownErrorType**

  Unknown error.
- class **crypto::stringTooLarge**

  String size error.
- class **crypto::errorListener**

  crypto::error *(p. 79)* *listener*
- class **crypto::errorSender**

  *Sends **crypto::error*** *(p. 79).*

Namespaces

- **crypto**

Typedefs

- typedef os::smart_ptr< error > **crypto::errorPointer**

  *Smart pointer to **crypto::error** (p. 79).*

3.15.1 Detailed Description

Declaration of cryptographic errors.

Author

Jonathan Bedard

Date

4/1/2016

Bug None

Declares a number of errors for the CryptoGateway package. Also declares two classes to manage the sending and listening for the throwing of **crypto::errorPointer** (p. 50).

3.16 cryptoFileTest.cpp File Reference

Implementation for cryptographic file testing.

3.16.1 Detailed Description

Implementation for cryptographic file testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file implements a series of tests designed to confirm the stability of cryptographic save file and load file functions.

3.17 cryptoFileTest.h File Reference

Header for cryptographic file testing.

3.17.1 Detailed Description

Header for cryptographic file testing.

Author

Jonathan Bedard

Date

3/5/2016

Bug No known bugs.

This contains a number of test suites and supporting classes which are designed to test the functionality of saving and loading cryptographic files, both binary and EXML.

3.18 CryptoGateway.h File Reference

Global include file.

Namespaces

- **crypto**

Variables

- bool **crypto::global_logging**
  Deprecated logging flag.

3.18.1 Detailed Description

Global include file.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

This file contains all of the headers in the CryptoGateway library. Project which depend on the CryptoGateway library need only include this file.

3.19 cryptoHash.cpp File Reference

Implementation of crypto hashing.

3.19.1 Detailed Description

Implementation of crypto hashing.
Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements basic hashing frameworks and the XOR hash. Note that the XOR hash is not cryptographically secure. Consult **cryptoHash.h** (p. 24) for details.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure. Consult the **RC4_Hash.h** (p. 37) for details.

3.20 cryptoHash.h File Reference

Declaration of crypto hashing.

Classes

- class **crypto::hash**
  Base hash class.
- class **crypto::xorHash**
  XOR hash class.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const hash &num)
  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, hash &num)
  Input stream operator.
- template<class hashClass >
  hashClass **crypto::hashData** (uint16_t hashType, const unsigned char *data, uint32_t length)
  Hashes data with the specified algorithm.

3.20.1 Detailed Description

Declaration of crypto hashing.
Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares base cryptographic hashing class and functions. All hash algorithms should extend this hash class.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure.

3.21 cryptoLogging.cpp File Reference

Logging for crypto namespace, implementation.

3.21.1 Detailed Description

Logging for crypto namespace, implementation.
Jonathan Bedard

Date

2/23/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the crypto namespace.

3.22 cryptoLogging.h File Reference

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::cryptoout_func** ()
  Standard out object for crypto namespace.
- std::ostream & **crypto::cryptoerr_func** ()
  Standard error object for crypto namespace.

Variables

- os::smart_ptr< std::ostream > **crypto::cryptoout_ptr**
  Standard out pointer for crypto namespace.
- os::smart_ptr< std::ostream > **crypto::cryptoerr_ptr**
  Standard error pointer for crypto namespace.

3.23 cryptoNumber.cpp File Reference

Implements basic number types.

3.23.1 Detailed Description

Implements basic number types.

Author

Jonathan Bedard

Date

4/3/2016

Bug No known bugs.

Implements basic large numbers and the more specific large integer. Consult **cryptoNumber.h** (p. 26) for details.

3.24 cryptoNumber.h File Reference

Defines basic number types.

Classes

- class **crypto::number**

  Basic number definition.
- class **crypto::integer**

  Integer number definition.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const number &num)

  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, number &num)

  Input stream operator.

3.24.1 Detailed Description

Defines basic number types.

Author

Jonathan Bedard

Date

3/2/2016

Bug No known bugs.

Contains declarations of large numbers for usage inside the CryptoGateway. The two numbers defined in this file are the general structure for large numbers and a basic integer.

3.25 cryptoNumberTest.cpp File Reference

Testing **crypto::number** (p. 173) and **crypto::integer** (p. 125).

3.25.1 Detailed Description

Testing **crypto::number** (p. 173) and **crypto::integer** (p. 125).

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file has a series of tests which confirm the functionality of **crypto::integer** (p. 125) and it's base class, **crypto::number** (p. 173).

3.26 cryptoPublicKey.cpp File Reference

Generalized and RSA public key implementation.

3.26.1 Detailed Description

Generalized and RSA public key implementation.

Author

Jonathan Bedard

Date

4/3/2016

Bug No known bugs.

Contains implementation of the generalized public key and the RSA public key. Consult **crypto↩ PublicKey.h** (p. 28) for details.

3.27 cryptoPublicKey.h File Reference

Generalized and RSA public keys.

Classes

- class **crypto::keyChangeReceiver**

  Interface for receiving key changes.
- class **crypto::keyChangeSender**

  *Interface inherited by **publicKey** (p. 195).*
- class **crypto::publicKey**

  Base public-key class.
- class **crypto::publicRSA**

  RSA public-key encryption.
- class **crypto::RSAKeyGenerator**

  Helper key generation class.

Namespaces

- **crypto**

3.27.1 Detailed Description

Generalized and RSA public keys.

Author

Jonathan Bedard

Date

3/19/2016

Bug No known bugs.

Contains declarations of the generalized public key and the RSA public key. These classes can both encrypt and decrypt public keys.

3.28 cryptoTest.cpp File Reference

CryptoGateway library test constructor.

3.28.1 Detailed Description

CryptoGateway library test constructor.

Author

Jonathan Bedard

Date

4/7/2016

Bug No known bugs.

Binds all test suites for the test::CryptoGatewayLibraryTest. This library test is called "Crypto↩ Gateway."

3.29 cryptoTest.h File Reference

CryptoGateway library test header.

3.29.1 Detailed Description

CryptoGateway library test header.

Author

Jonathan Bedard

Date

4/2/2016

Bug No known bugs.

Contains declarations need to bind the CryptoGateway test library to the unit test driver.

3.30 gateway.cpp File Reference

Implements the gateway.

3.30.1 Detailed Description

Implements the gateway.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the gateway defined in **gateway.h** (p. 30). Consult **gateway.h** (p. 30) for details.

3.31 gateway.h File Reference

Defines the gateway.

Classes

- class **crypto::gatewaySettings**
  Holds settings for gateway encryption.
- class **crypto::gateway**
  Security gateway.

Namespaces

- **crypto**

3.31.1 Detailed Description

Defines the gateway.

Author

Jonathan Bedard

Date

4/17/2016

Bug No known bugs.

This file contains the declaration for the gateway and the gateway settings. This header file is the culmination of the CryptoGateway library.

Note that due to development constraints, the gatewaySettings class is being pushed out in a frame-work form and is intended to contain a large set of algorithm definitions as well as an algorithm use agreement protocol.

3.32 gatewayTest.cpp File Reference

Implementation for end-to-end gateway testing.

3.32.1 Detailed Description

Implementation for end-to-end gateway testing.

Author

Jonathan Bedard

Date

4/3/2016

Bug No known bugs.

This file contains implementation of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

3.33 gatewayTest.h File Reference

Header for end-to-end gateway testing.

3.33.1 Detailed Description

Header for end-to-end gateway testing.

Author

Jonathan Bedard

Date

3/20/2016

Bug No known bugs.

This header contains declarations of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

3.34 hashTest.cpp File Reference

Implementation for hash tests.

3.34.1 Detailed Description

Implementation for hash tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic hash testing. These tests confirm that the respective hash algorithms are outputting their expected value.

3.35 hashTest.h File Reference

Header for hash testing.

3.35.1 Detailed Description

Header for hash testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic hash algorithms.

3.36 hexConversion.cpp File Reference

Hex conversion implementation.

3.36.1 Detailed Description

Hex conversion implementation.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Implements the set of hex conversion functions. Consult **hexConversion.h** (p. 33) for details.

3.37 hexConversion.h File Reference

Hex conversion header.

Namespaces

- **crypto**

Functions

- bool **crypto::isHexCharacter** (char c)
  Check the character type.
- std::string **crypto::toHex** (unsigned char i)
  Converts an 8 bit integer to a hex string.
- std::string **crypto::toHex** (uint32_t i)
  Converts an 32 bit integer to a hex string.
- unsigned char **crypto::fromHex8** (const std::string &str)
  Converts a hex string to an 8 bit integer.
- uint32_t **crypto::fromHex32** (const std::string &str)
  Converts a hex string to an 32 bit integer.

3.37.1 Detailed Description

Hex conversion header.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Contains a set of functions to convert integers and characters from a hex string and converts hex strings to integers and characters.

3.38 keyBank.cpp File Reference

Implimentation for the AVL tree based key bank.

3.38.1 Detailed Description

Implimentation for the AVL tree based key bank.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contians the implimentation for the **crypto::avlKeyBank** (p. 57) and supporting classes. Consult **keyBank.h** (p. 34) for details.

3.39 keyBank.h File Reference

Header for the AVL tree based key bank.

Classes

- class **crypto::nodeGroup**

  Node group.
- class **crypto::nodeNameReference**

  Name storage node.
- class **crypto::nodeKeyReference**

  Key storage node.
- class **crypto::keyBank**

  Key bank interface.
- class **crypto::avlKeyBank**

  AVL key back.

Namespaces

- **crypto**

3.39.1 Detailed Description

Header for the AVL tree based key bank.

Author

Jonathan Bedard

Date

3/19/2016

Bug No known bugs.

This file contians declarations for the **crypto::avlKeyBank** (p. 57) and supporting classes. Note that the key-bank may later be implimented with more advanced datastructures.

3.40 message.cpp File Reference

Crypto-Gateway message implementation.

3.40.1 Detailed Description

Crypto-Gateway message implementation.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

Implements the message used by the crypto-gateway to pass encrypted data between machines.

3.41 message.h File Reference

Crypto-Gateway message.

Classes

- class **crypto::message**
  Crypto-Gateway message.

Namespaces

- **crypto**

3.41.1 Detailed Description

Crypto-Gateway message.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

The message declared in this file acts as a message for the Crypto-Gateway. These messages are intended to be converted to machine-to-machine communication.

3.42 publicKeyPackage.cpp File Reference

Namespaces

- **crypto**

Variables

- static os::smart_ptr< publicKeyTypeBank > **crypto::_singleton**

3.43 publicKeyPackage.h File Reference

Classes

- class **crypto::publicKeyPackageFrame**
- class **crypto::publicKeyPackage**< **pkType** >
- class **crypto::publicKeyTypeBank**

Namespaces

- **crypto**

3.44 publicKeyTest.h File Reference

Public Key tests.

3.44.1 Detailed Description

Public Key tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Since the public key tests are defined by very simple tests, the template testing classes contained in this file are also defined in this file. There is no .cpp file paired with this particular header.

3.45 RC4_Hash.cpp File Reference

3.46 RC4_Hash.h File Reference

Classes

- class **crypto::rc4Hash**
  RC-4 hash class.

Namespaces

- **crypto**

3.47 staticTestKeys.cpp File Reference

Auto-generated.

3.47.1 Detailed Description

Auto-generated.

Author

None

Bug None

3.48 staticTestKeys.h File Reference

Auto-generated.

3.48.1 Detailed Description

Auto-generated.

Author

None

Bug None

3.49 streamCipher.cpp File Reference

3.50 streamCipher.h File Reference

Classes

- class **crypto::streamCipher**
- class **crypto::RCFour**
- class **crypto::streamPacket**
- class **crypto::streamEncrypter**
- class **crypto::streamDecrypter**

Namespaces

- **crypto**

Variables

- bool **global_logging**

3.50.1 Variable Documentation

bool global_logging

3.51 streamPackage.cpp File Reference

Namespaces

- **crypto**

Variables

- static os::smart_ptr< streamPackageTypeBank > **crypto::_singleton**

3.52 streamPackage.h File Reference

Classes

- class **crypto::streamPackageFrame**
- class **crypto::streamPackage**< **streamType, hashType** >
- class **crypto::streamPackageTypeBank**

Namespaces

- **crypto**

3.53 streamTest.cpp File Reference

Implementation for stream tests.

3.53.1 Detailed Description

Implementation for stream tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic stream testing. These tests confirm that the respective stream algorithms are outputting their expected value.

3.54 streamTest.h File Reference

Header for stream testing.

3.54.1 Detailed Description

Header for stream testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic stream objects.

3.55 testKeyGeneration.cpp File Reference

3.56 testKeyGeneration.h File Reference

Implementation of test key binding.

3.56.1 Detailed Description

Implementation of test key binding.
Binds generated testing keys.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the binding of the static test keys to arrays in memory. Consult **testKeyGeneration.h** (p. 40) for details.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Provides access to the keys generated and stored in **staticTestKeys.h** (p. 37) and **staticTest↩ Keys.cpp** (p. 37). These keys are always copied into a raw array of uint32_t.

3.57 user.cpp File Reference

Implementation of the CryptoGateway user.

3.57.1 Detailed Description

Implementation of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/18/2016

Bug None

Provides an implementation of user which has a user-name, password and associated bank of public keys. Consult **user.h** (p. 41) for details.

3.58 user.h File Reference

Definition of the CryptoGateway user.

Classes

- class **crypto::user**
  Primary user class.

Namespaces

- **crypto**

3.58.1 Detailed Description

Definition of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/14/2016

Bug None

Provides a definition of user which has a user-name, password and associated bank of public keys.

3.59 XMLEncryption.cpp File Reference

Namespaces

- **crypto**

Functions

- static std::vector< std::string > **crypto::generateArgumentList** (os::smartXMLNode head)
- static void **crypto::recursiveXMLPrinting** (os::smartXMLNode head, os::smart_ptr< stream↩ Cipher > strm, std::vector< std::string > args, std::ofstream &ofs)
- static os::smartXMLNode **crypto::recursiveXMLBuilding** (os::smart_ptr< streamCipher > strm, std::vector< std::string > args, std::ifstream &ifs)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< streamPackageFrame > spf)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, unsigned char ∗sym↩ Key, unsigned int passwordLength, os::smart_ptr< streamPackageFrame > spf)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< public↩ Key > pbk, unsigned int lockType, os::smart_ptr< streamPackageFrame > spf)

- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< number > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< streamPackage↩ Frame > spf)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, unsigned char ∗symKey, unsigned int passwordLength)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)

3.60 XMLEncryption.h File Reference

Namespaces

- **crypto**

Functions

- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, unsigned char ∗sym↩ Key, unsigned int passwordLength, os::smart_ptr< streamPackageFrame > spf)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< streamPackageFrame > spf)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< public↩ Key > pbk, unsigned int lockType, os::smart_ptr< streamPackageFrame > spf)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< number > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< streamPackage↩ Frame > spf)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, unsigned char ∗symKey, unsigned int passwordLength)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 crypto Namespace Reference

Classes

- class **actionOnFileClosed**

  File closed error.
- class **actionOnFileError**

  File error.
- class **avlKeyBank**

  AVL key back.
- class **binaryDecryptor**

  Encrypted binary file output.
- class **binaryEncryptor**

  Encrypted binary file output.
- class **bufferLargeError**

  Buffer too large.
- class **bufferSmallError**

  Buffer too small.
- class **customError**

  *Custom **crypto::error*** (p. *79*)*.*
- class **error**

  Sortable exception.
- class **errorListener**

  crypto::error (p. *79*) *listener*
- class **errorSender**

  *Sends **crypto::error*** (p. *79*)*.*
- class **fileFormatError**

  File format error.
- class **fileOpenError**

  File open error.

- class **gateway**

  Security gateway.
- class **gatewaySettings**

  Holds settings for gateway encryption.
- class **hash**

  Base hash class.
- class **hashCompareError**

  Hash mis-match.
- class **hashGenerationError**

  Hash generation error.
- class **illegalAlgorithmBind**

  Algorithm bound failure.
- class **insertionFailed**

  ADS Insertion Failed.
- class **integer**

  Integer number definition.
- class **keyBank**

  Key bank interface.
- class **keyChangeReceiver**

  Interface for receiving key changes.
- class **keyChangeSender**

  *Interface inherited by **publicKey** (p. 195).*
- class **keyMissing**

  Key missing error.
- class **masterMismatch**

  Master mis-match.
- class **message**

  Crypto-Gateway message.
- class **nodeGroup**

  Node group.
- class **nodeKeyReference**

  Key storage node.
- class **nodeNameReference**

  Name storage node.
- class **NULLDataError**

  NULL data error.
- class **NULLMaster**

  NULL master error.
- class **NULLPublicKey**

  NULL public-key error.
- class **number**

  Basic number definition.
- class **passwordLargeError**

Symmetric key too big.

- class **passwordSmallError**

  Symmetric key too small.
- class **publicKey**

  Base public-key class.
- class **publicKeyPackage**
- class **publicKeyPackageFrame**
- class **publicKeySizeWrong**

  Public-key size error.
- class **publicKeyTypeBank**
- class **publicRSA**

  RSA public-key encryption.
- class **rc4Hash**

  RC-4 hash class.
- class **RCFour**
- class **RSAKeyGenerator**

  Helper key generation class.
- class **streamCipher**
- class **streamDecrypter**
- class **streamEncrypter**
- class **streamPackage**
- class **streamPackageFrame**
- class **streamPackageTypeBank**
- class **streamPacket**
- class **stringTooLarge**

  String size error.
- class **unknownErrorType**

  Unknown error.
- class **user**

  Primary user class.
- class **xorHash**

  XOR hash class.

Typedefs

- typedef os::smart_ptr< **error** > **errorPointer**

  Smart pointer to ***crypto::error*** (p. 79).

Functions

- std::ostream & **operator**<< (std::ostream &os, const **hash** &num)

  Output stream operator.
- std::istream & **operator**>> (std::istream &is, **hash** &num)

  Input stream operator.
- template<class hashClass >
  hashClass **hashData** (uint16_t hashType, const unsigned char *data, uint32_t length)

  Hashes data with the specified algorithm.
- std::ostream & **cryptoout_func** ()

  Standard out object for crypto namespace.
- std::ostream & **cryptoerr_func** ()

  Standard error object for crypto namespace.
- std::ostream & **operator**<< (std::ostream &os, const **number** &num)

  Output stream operator.
- std::istream & **operator**>> (std::istream &is, **number** &num)

  Input stream operator.
- bool **isHexCharacter** (char c)

  Check the character type.
- std::string **toHex** (unsigned char i)

  Converts an 8 bit integer to a hex string.
- std::string **toHex** (uint32_t i)

  Converts an 32 bit integer to a hex string.
- unsigned char **fromHex8** (const std::string &str)

  Converts a hex string to an 8 bit integer.
- uint32_t **fromHex32** (const std::string &str)

  Converts a hex string to an 32 bit integer.
- static std::vector< std::string > **generateArgumentList** (os::smartXMLNode head)
- static void **recursiveXMLPrinting** (os::smartXMLNode head, os::smart_ptr< **streamCipher** > strm, std::vector< std::string > args, std::ofstream &ofs)
- static os::smartXMLNode **recursiveXMLBuilding** (os::smart_ptr< **streamCipher** > strm, std::vector< std::string > args, std::ifstream &ifs)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< **streamPackageFrame** > spf)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, unsigned char *symKey, unsigned int passwordLength, os::smart_ptr< **streamPackageFrame** > spf)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< **publicKey** > pbk, unsigned int lockType, os::smart_ptr< **streamPackageFrame** > spf)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< **number** > **publicKey**, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > spf)
- os::smartXMLNode **EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **EXML_Input** (std::string path, unsigned char *symKey, unsigned int passwordLength)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **publicKey** > pbk, os::smart_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > &author)

- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **publicKey** > pbk)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **keyBank** > kyBank)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **keyBank** > kyBank, os↩ ::smart_ptr< **nodeGroup** > &author)

Variables

- bool **global_logging**

  Deprecated logging flag.
- os::smart_ptr< std::ostream > **cryptoout_ptr**

  Standard out pointer for crypto namespace.
- os::smart_ptr< std::ostream > **cryptoerr_ptr**

  Standard error pointer for crypto namespace.
- static os::smart_ptr< **publicKeyTypeBank** > **_singleton**
- static os::smart_ptr< **streamPackageTypeBank** > **_singleton**

5.1.1 Typedef Documentation

typedef os::smart_ptr<**error**> **crypto::errorPointer**

Smart pointer to **crypto::error** (p. 79).

5.1.2 Function Documentation

std::ostream& crypto::cryptoerr_func ( )

Standard error object for crypto namespace.
#define statements allow the user to call this function with "crypto::cryptoerr." Logging is achieved by using "crypto::cryptoerr" as one would use "std::cerr."

std::ostream& crypto::cryptoout_func ( )

Standard out object for crypto namespace.
#define statements allow the user to call this function with "crypto::cryptoout." Logging is achieved by using "crypto::cryptoout" as one would use "std::cout."

os::smartXMLNode crypto::EXML_Input (std::string path, std::string password)

os::smartXMLNode crypto::EXML_Input (std::string path, unsigned char * symKey, unsigned int passwordLength)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **publicKey** > pbk, os::smart_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > & author)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **publicKey** > pbk)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **keyBank** > kyBank)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > & author)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< **streamPackageFrame** > spf)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, unsigned char * symKey, unsigned int passwordLength, os::smart_ptr< **streamPackageFrame** > spf)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, os::smart_ptr< **publicKey** > pbk, unsigned int lockType, os::smart_ptr< **streamPackageFrame** > spf)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, os::smart_ptr< **number** > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > spf)

uint32_t crypto::fromHex32 (const std::string & str)

Converts a hex string to an 32 bit integer.

Parameters

`in`	*str*	Hex string to convert

Returns

str converted to integer

unsigned char crypto::fromHex8 (const std::string & str)

Converts a hex string to an 8 bit integer.

Parameters

`in`	*str*	Hex string to convert

Returns

str converted to integer

static std::vector<std::string> crypto::generateArgumentList (os::smartXMLNode head) `[static]`

template<class hashClass > hashClass crypto::hashData (uint16_t hashType, const unsigned char * data, uint32_t length)

Hashes data with the specified algorithm.

Hashes the provided data array returning a hash of the specified algorithm. This is a template function, which calls the static hash function for the specified algorithm.

Parameters

in	*hashType*	Size of hash
in	*data*	Data array to be hashed
in	*length*	Length of data to be hashed

Returns

Hash for data array

bool crypto::isHexCharacter (char c)

Check the character type.

Checks if the character is a valid hex character. That is, 0-9 and A-F.

Parameters

in	*c*	Character to test

Returns

true if a hex character, else, false

std::ostream& crypto::operator<< (std::ostream & os, const **number** & num)

Output stream operator.

Parameters

	[in/out]	os Output stream
in	*num*	Number to be output

Returns

reference to std::ostream& os

std::ostream& crypto::operator<< (std::ostream & os, const **hash** & num)

Output stream operator.

Outputs a hex version of the hash to the provided output stream. This output will look identical for two hashes which are equal but have different algorithms.

Parameters

	[in/out]	os Output stream
in	*num*	Hash to be printed return Reference to output stream

std::istream& crypto::operator>> (std::istream & is, **number** & num)

Input stream operator.

Parameters

	[in/out]	is Input stream
in	*num*	Number to set with the string

Returns

reference to std::istream& is

std::istream& crypto::operator>> (std::istream & is, **hash** & num)

Input stream operator.

Inputs a hex version of the hash from the provided output stream. This function must receive a constructed hash, although it will rebuild the provided hash with the stream data.

Parameters

	[in/out]	is Input stream
in	*num*	Hash to be created return Reference to input stream

static os::smartXMLNode crypto::recursiveXMLBuilding (os::smart_ptr< **streamCipher** > strm, std::vector< std::string > args, std::ifstream & ifs) [static]

static void crypto::recursiveXMLPrinting (os::smartXMLNode head, os::smart_ptr< **streamCipher** > strm, std::vector< std::string > args, std::ofstream & ofs) [static]

std::string crypto::toHex (unsigned char i)

Converts an 8 bit integer to a hex string.

Parameters

in	*i*	Integer to convert

Returns

i converted to hex string

std::string crypto::toHex (uint32_t i)

Converts an 32 bit integer to a hex string.

Parameters

in	*i*	Integer to convert

Returns

i converted to hex string

5.1.3 Variable Documentation

os::smart_ptr<**publicKeyTypeBank**> crypto::_singleton [static]

os::smart_ptr<**streamPackageTypeBank**> crypto::_singleton [static]

os::smart_ptr<std::ostream> crypto::cryptoerr_ptr

Standard error pointer for crypto namespace.

This std::ostream is used as standard error for the crypto namespace. This pointer can be swapped out to programmatically redirect standard error for the crypto namespace.

os::smart_ptr<std::ostream> crypto::cryptoout_ptr

Standard out pointer for crypto namespace.

This std::ostream is used as standard out for the crypto namespace. This pointer can be swapped out to programmatically redirect standard out for the crypto namespace.

bool crypto::global_logging

Deprecated logging flag.

Old logging flag. Deprecated in the new CryptoGateway files. This has been replaced by the logging system outlined in this file.

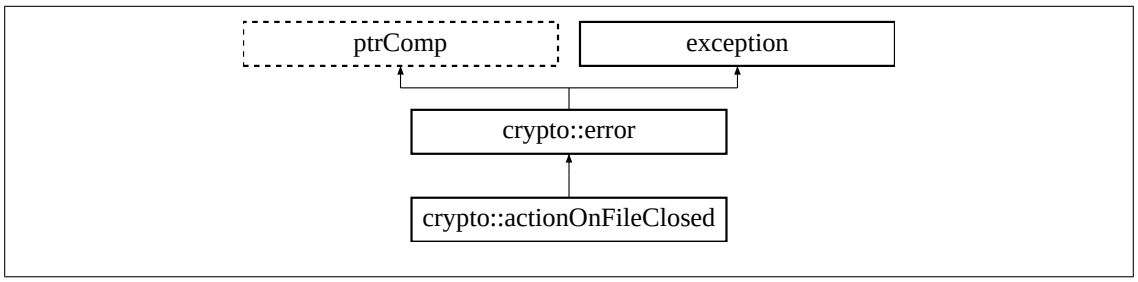
Chapter 6

Class Documentation

6.1 crypto::actionOnFileClosed Class Reference

File closed error.

Inheritance diagram for crypto::actionOnFileClosed:



Public Member Functions

- virtual ~**actionOnFileClosed** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Action on File Closed".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot preform action on a file in the closed state".

6.1.1 Detailed Description

File closed error.

Thrown when an action is attempted on a file which is already closed.

6.1.2 Constructor & Destructor Documentation

virtual crypto::actionOnFileClosed::~actionOnFileClosed ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.1.3 Member Function Documentation

std::string crypto::actionOnFileClosed::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot preform action on a file in the closed state".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::actionOnFileClosed::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Action on File Closed".

Returns

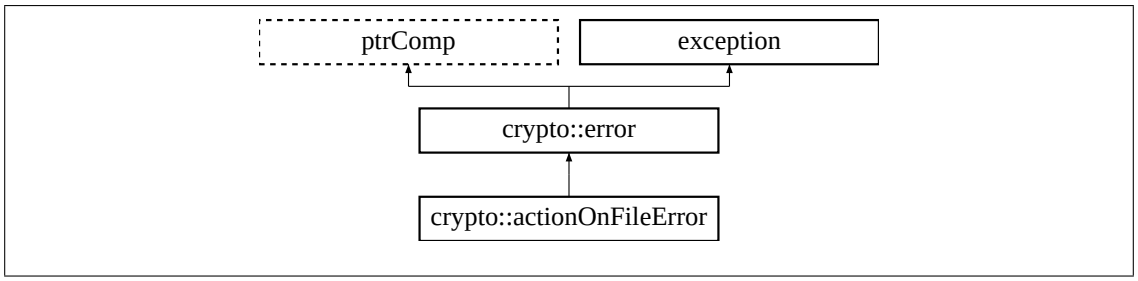
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.2 crypto::actionOnFileError Class Reference

File error.

Inheritance diagram for crypto::actionOnFileError:



Public Member Functions

- virtual ~**actionOnFileError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Action on File Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot preform action on a file in the error state".

6.2.1 Detailed Description

File error.

Thrown when an action is attempted on a file in the error state.

6.2.2 Constructor & Destructor Documentation

virtual crypto::actionOnFileError::~actionOnFileError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.2.3 Member Function Documentation

std::string crypto::actionOnFileError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot preform action on a file in the error state".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::actionOnFileError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Action on File Error".

Returns

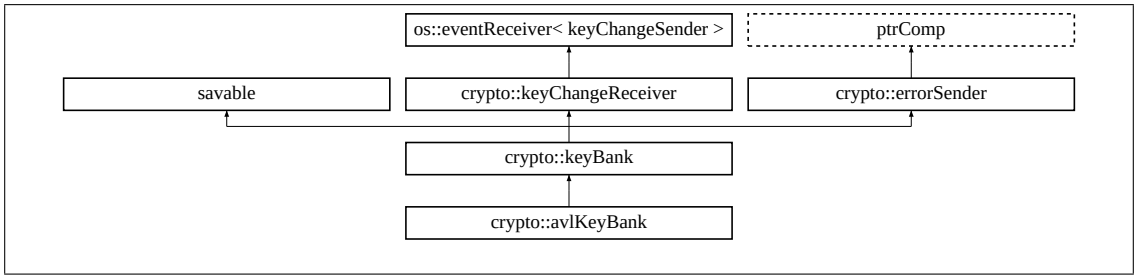
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.3 crypto::avlKeyBank Class Reference

AVL key back.

Inheritance diagram for crypto::avlKeyBank:



Public Member Functions

- **avlKeyBank** (std::string **savePath**="", const unsigned char *key=NULL, unsigned int keyLen=0, os::smart_ptr< **streamPackageFrame** > strmPck=NULL)

  Construct with save path.
- virtual ~**avlKeyBank** ()

  Virtual destructor.
- void **save** ()

  Saves bank to file.
- os::smart_ptr< **nodeGroup** > **addPair** (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Adds authenticated node to bank.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeNameReference** > name)

  Find by group name reference.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeKeyReference** > key)

  Find by group key reference.
- os::smart_ptr< **nodeGroup** > **find** (std::string groupName, std::string name)

  Find by group name and name.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Find by key information.

Protected Member Functions

- void **pushNewNode** (os::smart_ptr< **nodeNameReference** > name)

  Add name node.
- void **pushNewNode** (os::smart_ptr< **nodeKeyReference** > key)

  Add key node.
- void **load** ()

  Loads bank from file.

Private Attributes

- os::asyncAVLTree< **nodeNameReference** > **nameTree**

  List of all names associated with this node.
- os::asyncAVLTree< **nodeKeyReference** > **keyTree**

  List of all keys associated with this node.
- os::asyncAVLTree< **nodeGroup** > **nodeBank**

  List of all node groups.

Additional Inherited Members

6.3.1 Detailed Description

AVL key back.

The AVL key bank stores keys in a series of AVL trees. All keys in the bank are loaded into memory when the file is loaded, meaning that there is a limited number of keys that can be practically managed through an AVL key bank.

6.3.2 Constructor & Destructor Documentation

crypto::avlKeyBank::avlKeyBank (std::string savePath = "", const unsigned char * key = NULL, unsigned int keyLen = 0, os::smart_ptr< **streamPackageFrame** > strmPck = NULL)

Construct with save path.

Intializes the key bank and loads the the bank from a file.

Parameters

in	*savePath*	Path to save file, empty by default
in	*key*	Symetric key
in	*keyLen*	Length of symetric key
in	*strmPck*	Definition of algorithms used

virtual crypto::avlKeyBank::~avlKeyBank ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.3.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::addPair (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [virtual]

Adds authenticated node to bank.

Note that if a node has not be authenticated, adding it to the bank will cause a potential security vulnerability. Nodes should be authenticated before being added to the bank.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node
in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Return reference to the new node group

Implements **crypto::keyBank** (p. 137).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **nodeNameReference** > name) [virtual]

Find by group name reference.

Parameters

in	*name*	Name reference to be searched

Returns

Node group found by arguments

Implements **crypto::keyBank** (p. 138).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **nodeKeyReference** > key) [virtual]

Find by group key reference.

Parameters

in	*key*	Key reference to be searched

Returns

Node group found by arguments

Implements **crypto::keyBank** (p. 138).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (std::string groupName, std::string name) [inline], [virtual]

Find by group name and name.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node

Returns

Node group found by arguments

Reimplemented from **crypto::keyBank** (p. 138).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [inline], [virtual]

Find by key information.

Parameters

in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Node group found by arguments

Reimplemented from **crypto::keyBank** (p. 139).

void crypto::avlKeyBank::load ( ) [protected], [virtual]

Loads bank from file.

Returns

void

Implements **crypto::keyBank** (p. 139).

void crypto::avlKeyBank::pushNewNode (os::smart_ptr< **nodeNameReference** > name) [protected], [virtual]

Add name node.

Inserts a name node into the bank. The name node has a reference to a node group.

Parameters

in	*name*	Name node to be added

Returns

void

Implements **crypto::keyBank** (p. 139).

void crypto::avlKeyBank::pushNewNode (os::smart_ptr< **nodeKeyReference** > key) [protected], [virtual]

Add key node.

Inserts a key node into the bank. The key node has a reference to a node group.

Parameters

in	*key*	Key node to be added

Returns

void

Implements **crypto::keyBank** (p. 140).

void crypto::avlKeyBank::save ( ) [virtual]

Saves bank to file.

Returns

void

Implements **crypto::keyBank** (p. 140).

6.3.4 Member Data Documentation

os::asyncAVLTree<**nodeKeyReference**> crypto::avlKeyBank::keyTree [private]

List of all keys associated with this node.

os::asyncAVLTree<**nodeNameReference**> crypto::avlKeyBank::nameTree [private]

List of all names associated with this node.

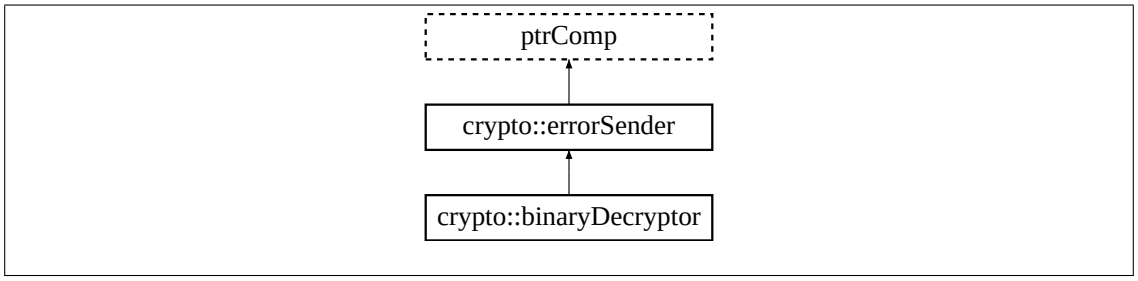
os::asyncAVLTree<**nodeGroup**> crypto::avlKeyBank::nodeBank [private]

List of all node groups.

6.4 crypto::binaryDecryptor Class Reference

Encrypted binary file output.

Inheritance diagram for crypto::binaryDecryptor:



Public Member Functions

- **binaryDecryptor** (std::string file_name, os::smart_ptr< **keyBank** > kBank)
  Construct with public key.
- **binaryDecryptor** (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock)
  Construct with public key.
- **binaryDecryptor** (std::string file_name, std::string password)
  Construct with password.
- **binaryDecryptor** (std::string file_name, unsigned char *key, unsigned int keyLen)
  Construct with symmetric key.
- unsigned char **read** ()
  Attempts to read a single character.
- unsigned int **read** (unsigned char *data, unsigned int dataLen)
  Attempts to read a block of data.
- void **close** ()
  Closes the output file.
- const std::string & **fileName** () const
  Returns the name of target file.
- const os::smart_ptr< **streamPackageFrame** > **streamAlgorithm** () const
  Returns the stream algorithm definition.
- bool **good** () const
  Returns the current file state.
- bool **finished** () const
  Returns if the file has finished writing.
- unsigned long **bytesLeft** () const
  Returns the number of bytes left in the file.
- os::smart_ptr< **nodeGroup** > **author** ()
  Pointer to the user which signed this file.
- virtual ~**binaryDecryptor** ()
  Virtual destructor.

Private Member Functions

- void **build** (unsigned char *key=NULL, unsigned int keyLen=0)
  Central constructor function.

Private Attributes

- os::smart_ptr< **publicKey** > **_publicKeyLock**

  Pointer to the optional public key.
- os::smart_ptr< **keyBank** > **_keyBank**

  Pointer to the key bank (to confirm public keys)
- os::smart_ptr< **nodeGroup** > **_author**

  Pointer to the user which signed this file.
- os::smart_ptr< **streamPackageFrame** > **_streamAlgorithm**

  Pointer to the mandatory stream algorithm definition.
- os::smart_ptr< **streamCipher** > **currentCipher**

  Pointer to the current stream cipher.
- bool **_state**

  State of the output file.
- bool **_finished**

  Has the file been closed.
- std::string **_fileName**

  Name of the file being read from.
- std::ifstream **input**

  Binary input file.
- unsigned long **_bytesLeft**

  Number of bytes left in the file.

Additional Inherited Members

6.4.1 Detailed Description

Encrypted binary file output.

The user defines an encryption algorithm and key, then places data into the file. This data is automatically encrypted with the specified algorithm and key.

6.4.2 Constructor & Destructor Documentation

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, os::smart_ptr< **keyBank** > kBank)

Construct with public key.

Constructs the file reader with a public key.

Parameters

`in`	*file_name*	Name of input file
`in`	*kBank*	Record of public keys

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock)

Construct with public key.

Constructs the file reader with a public key.

Parameters

in	*file_name*	Name of input file
in	*publicKeyLock*	Public key to decrypt data

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, std::string password)

Construct with password.

Constructs the file reader with a password.

Parameters

in	*file_name*	Name of input file
in	*password*	Password to decrypt data

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, unsigned char * key, unsigned int keyLen)

Construct with symmetric key.

Constructs the file reader with a symmetric key.

Parameters

in	*file_name*	Name of input file
in	*key*	Symmetric key byte array
in	*keyLen*	Size of the symmetric key

virtual crypto::binaryDecryptor::~binaryDecryptor ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Also closes the input file.

6.4.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::binaryDecryptor::author ( )

Pointer to the user which signed this file.

Returns

crypto::binaryDecryptor::_author (p. 67)

void crypto::binaryDecryptor::build (unsigned char * key = NULL, unsigned int keyLen = 0) [private]

Central constructor function.

This function reads the header of the encrypted binary file and attempts to initialize a stream cipher for decryption. Note that there is no guarantee that this can be done with the information given to the class. In this event, the class logs the error and sets it's state to false.

Parameters

in	*key*	Symmetric key, NULL by default
in	*keyLen*	Length of symmetric key, 0 by default

Returns

void

unsigned long crypto::binaryDecryptor::bytesLeft ( ) const [inline]

Returns the number of bytes left in the file.

Returns

crypto::binaryDecryptor::_bytesLeft (p. 67)

void crypto::binaryDecryptor::close ( )

Closes the output file.

Returns

void

const std::string& crypto::binaryDecryptor::fileName ( ) const [inline]

Returns the name of target file.

Returns

crypto::binaryDecryptor::_fileName (p. 68)

bool crypto::binaryDecryptor::finished ( ) const [inline]

Returns if the file has finished writing.

Returns

crypto::binaryDecryptor::_finished (p. 68)

bool crypto::binaryDecryptor::good ( ) const [inline]

Returns the current file state.

Returns

crypto::binaryDecryptor::_state (p. 68)

unsigned char crypto::binaryDecryptor::read ( )

Attempts to read a single character.

Note that if the reader is in a "good" state, then this function will read and decrypt a single byte of the file.

Returns

Character read, 0 if failed

unsigned int crypto::binaryDecryptor::read (unsigned char * data, unsigned int dataLen)

Attempts to read a block of data.

Note that if the reader is in a "good" state, then this function will read and decrypt the entire block of data requested.

Parameters

out	*data*	Array to place read data into
in	*dataLen*	Number of bytes attempting to read

Returns

Number of bytes read

const os::smart_ptr<**streamPackageFrame**> crypto::binaryDecryptor::streamAlgorithm ( ) const [inline]

Returns the stream algorithm definition.

Returns

crypto::binaryDecryptor::_streamAlgorithm (p. 68)

6.4.4 Member Data Documentation

os::smart_ptr<**nodeGroup**> crypto::binaryDecryptor::_author [private]

Pointer to the user which signed this file.

This is only populated if a key-bank is bound to the class.

unsigned long crypto::binaryDecryptor::_bytesLeft [private]

Number of bytes left in the file.

std::string crypto::binaryDecryptor::_fileName [private]

Name of the file being read from.

bool crypto::binaryDecryptor::_finished [private]

Has the file been closed.

If true, the file is closed. Else, the file is open and may be read from.

os::smart_ptr<**keyBank**> crypto::binaryDecryptor::_keyBank [private]

Pointer to the key bank (to confirm public keys)

os::smart_ptr<**publicKey**> crypto::binaryDecryptor::_publicKeyLock [private]

Pointer to the optional public key.

bool crypto::binaryDecryptor::_state [private]

State of the output file.

This state is either "good" or "bad." A bad file is not merely defined by crypto::binaryEncryptor↩ ::input, but also by any cryptographic abnormalities that are detected.

os::smart_ptr<**streamPackageFrame**> crypto::binaryDecryptor::_streamAlgorithm [private]

Pointer to the mandatory stream algorithm definition.

os::smart_ptr<**streamCipher**> crypto::binaryDecryptor::currentCipher [private]

Pointer to the current stream cipher.

The current cipher will be of the type defined in the algorithm definition. It will be initialized with either the provided public key or the provided password.

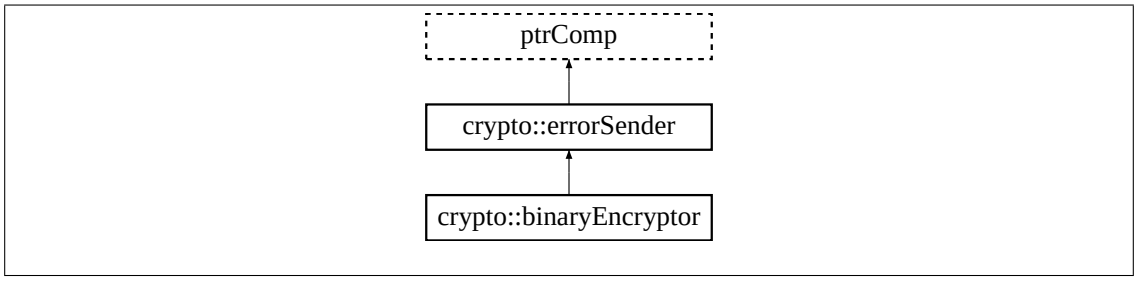
std::ifstream crypto::binaryDecryptor::input [private]

Binary input file.

6.5 crypto::binaryEncryptor Class Reference

Encrypted binary file output.

Inheritance diagram for crypto::binaryEncryptor:



Public Member Functions

- **binaryEncryptor** (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock, unsigned int lockType=file::PRIVATE_UNLOCK, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with public key.
- **binaryEncryptor** (std::string file_name, os::smart_ptr< **number** > **publicKey**, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with number and public key algorithm.
- **binaryEncryptor** (std::string file_name, std::string password, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with password.
- **binaryEncryptor** (std::string file_name, unsigned char *key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with symmetric key.
- void **write** (unsigned char data)

  Write a single character.
- void **write** (const unsigned char *data, unsigned int dataLen)

  Write an array of bytes.
- void **close** ()

  Closes the output file.
- const std::string & **fileName** () const

  Returns the name of target file.
- const os::smart_ptr< **streamPackageFrame** > **streamAlgorithm** () const

  Returns the stream algorithm definition.
- bool **good** () const

  Returns the current file state.
- bool **finished** () const

  Returns if the file has finished writing.
- virtual ~**binaryEncryptor** ()

  Virtual destructor.

Private Member Functions

- void **build** (unsigned char *key, unsigned int keyLen)

  Construct class with password.
- void **build** (os::smart_ptr< **publicKey** > publicKeyLock)

  Construct class with public key.
- void **build** (os::smart_ptr< **number** > pubKey, unsigned int pkAlgo, unsigned int pkSize)

  Construct class with number and algorithm.

Private Attributes

- unsigned int **_publicLockType**

  Defines method of locking the file.
- os::smart_ptr< **streamPackageFrame** > **_streamAlgorithm**

  Pointer to the mandatory stream algorithm definition.
- os::smart_ptr< **streamCipher** > **currentCipher**

  Pointer to the current stream cipher.
- bool **_state**

  State of the output file.
- bool **_finished**

  Has the file been closed.
- std::string **_fileName**

  Name of the file being written to.
- std::ofstream **output**

  Binary output file.

Additional Inherited Members

6.5.1 Detailed Description

Encrypted binary file output.

The user defines an encryption algorithm and key, then places data into the file. This data is automatically encrypted with the specified algorithm and key.

6.5.2 Constructor & Destructor Documentation

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock, unsigned int lockType = `file::PRIVATE_UNLOCK`, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with public key.

Constructs the file writer with a public key and an optional stream algorithm definition

Parameters

`in`	*file_name*	Name of output file
`in`	*publicKeyLock*	Public key to encrypt data

Parameters

in	*lockType*	Defines method of locking with public key
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, os::smart_ptr< **number** > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with number and public key algorithm.

Constructs the file writer with a public key and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	***publicKey*** *(p. 195)*	Number to encrypt data
in	*pkAlgo*	Defines public key algorithm
in	*pkSize*	Defines size of public key
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, std::string password, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with password.

Constructs the file writer with a password and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	*password*	String to encrypt data with
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with symmetric key.

Constructs the file writer with a symmetric key and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	*key*	Array of characters defining the symmetric key
in	*keyLen*	Length of symmetric key

Parameters

in	*stream_algo*	Optional stream algorithm definition

virtual crypto::binaryEncryptor::~binaryEncryptor ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Also closes the output file.

6.5.3 Member Function Documentation

void crypto::binaryEncryptor::build (unsigned char * key, unsigned int keyLen) [private]

Construct class with password.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple data formats to be converted into the key.

Parameters

in	*key*	Array of characters defining the symmetric key
in	*keyLen*	Length of symmetric key

Returns

void

void crypto::binaryEncryptor::build (os::smart_ptr< **publicKey** > publicKeyLock) [private]

Construct class with public key.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple types of data to be converted to a public key.

Parameters

in	*publicKeyLock*	Public key pair to encrypt data

Returns

void

void crypto::binaryEncryptor::build (os::smart_ptr< **number** > pubKey, unsigned int pkAlgo, unsigned int pkSize) [private]

Construct class with number and algorithm.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple types of data to be converted to a public key.

Parameters

in	*pubKey*	Public key to encrypt data
in	*pkAlgo*	Algorithm ID
in	*pkSize*	Size of public key

Returns

void

void crypto::binaryEncryptor::close ( )

Closes the output file.

Returns

void

const std::string& crypto::binaryEncryptor::fileName ( ) const [inline]

Returns the name of target file.

Returns

crypto::binaryEncryptor::_fileName (p. 74)

bool crypto::binaryEncryptor::finished ( ) const [inline]

Returns if the file has finished writing.

Returns

crypto::binaryEncryptor::_finished (p. 74)

bool crypto::binaryEncryptor::good ( ) const [inline]

Returns the current file state.

Returns

crypto::binaryEncryptor::_state (p. 74)

const os::smart_ptr<**streamPackageFrame**> crypto::binaryEncryptor::streamAlgorithm ( ) const [inline]

Returns the stream algorithm definition.

Returns

crypto::binaryEncryptor::_streamAlgorithm (p. 74)

void crypto::binaryEncryptor::write (unsigned char data)

Write a single character.

Parameters

in	*data*	Character to write

Returns

void

void crypto::binaryEncryptor::write (const unsigned char * data, unsigned int dataLen)

Write an array of bytes.

Parameters

in	*data*	Data array to write
in	*dataLen*	Length of data array

Returns

void

6.5.4 Member Data Documentation

std::string crypto::binaryEncryptor::_fileName [private]

Name of the file being written to.

bool crypto::binaryEncryptor::_finished [private]

Has the file been closed.
If true, the file is closed. Else, the file is open and may be written to.

unsigned int crypto::binaryEncryptor::_publicLockType [private]

Defines method of locking the file.

bool crypto::binaryEncryptor::_state [private]

State of the output file.
This state is either "good" or "bad." A bad file is not merely defined by **crypto::binaryEncryptor↩ ::output** (p. 75), but also by any cryptographic abnormalities that are detected.

os::smart_ptr<**streamPackageFrame**> crypto::binaryEncryptor::_streamAlgorithm [private]

Pointer to the mandatory stream algorithm definition.

os::smart_ptr<**streamCipher**> crypto::binaryEncryptor::currentCipher [private]

Pointer to the current stream cipher.

The current cipher will be of the type defined in the algorithm definition. It will be initialized with either the provided public key or the provided password.

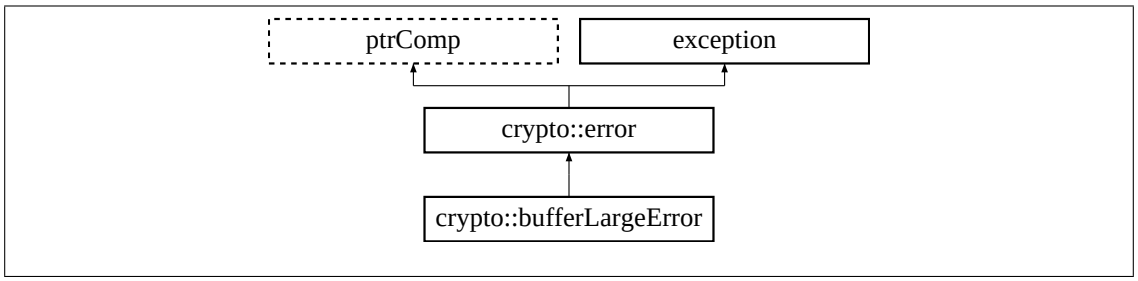
std::ofstream crypto::binaryEncryptor::output [private]

Binary output file.

6.6 crypto::bufferLargeError Class Reference

Buffer too large.

Inheritance diagram for crypto::bufferLargeError:



Public Member Functions

- virtual ~**bufferLargeError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Buffer Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Buffer too large".

6.6.1 Detailed Description

Buffer too large.

Thrown when the buffer provided to some cryptographic function is too large.

6.6.2 Constructor & Destructor Documentation

virtual crypto::bufferLargeError::~bufferLargeError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.6.3 Member Function Documentation

std::string crypto::bufferLargeError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Buffer too large".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::bufferLargeError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Buffer Size Error".

Returns

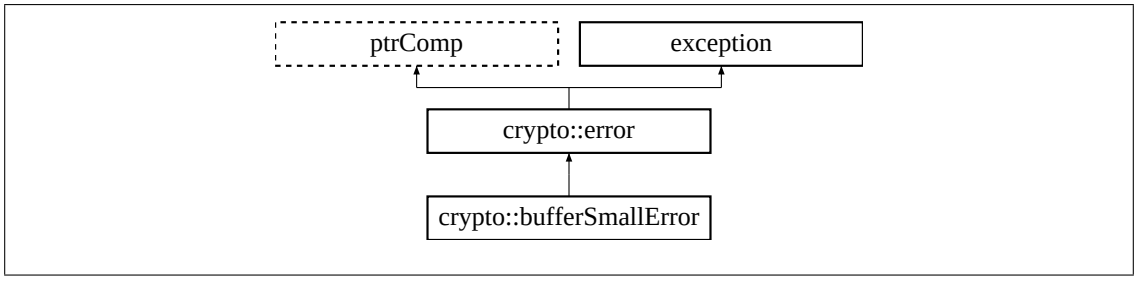
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.7 crypto::bufferSmallError Class Reference

Buffer too small.

Inheritance diagram for crypto::bufferSmallError:



Public Member Functions

- virtual ~**bufferSmallError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Buffer Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Buffer too small".

6.7.1 Detailed Description

Buffer too small.

Thrown when the buffer provided to some cryptographic function is too small.

6.7.2 Constructor & Destructor Documentation

virtual crypto::bufferSmallError::~bufferSmallError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.7.3 Member Function Documentation

std::string crypto::bufferSmallError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Buffer too small".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::bufferSmallError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Buffer Size Error".

Returns

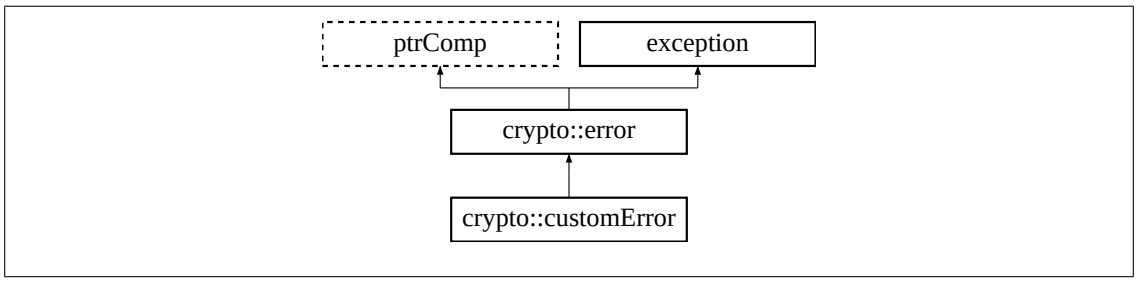
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.8 crypto::customError Class Reference

Custom **crypto::error** (p. 79).

Inheritance diagram for crypto::customError:



Public Member Functions

- **customError** (std::string name, std::string description)

  Custom error constructor.
- virtual ~**customError** () throw ()

  Virtual destructor.

- std::string **errorTitle** () const

  *Short error descriptor Returns "<name>" (**crypto::customError::_name** (p. 79))*
- std::string **errorDescription** () const

  *Long error descriptor Returns "<description>" (**crypto::customError::_description** (p. 79))*

Private Attributes

- std::string **_name**
- std::string **_description**

6.8.1 Detailed Description

Custom **crypto::error** (p. 79).

Allows the programmer to define an error unique to a specific situation.

6.8.2 Constructor & Destructor Documentation

crypto::customError::customError (std::string name, std::string description) `[inline]`

Custom error constructor.

Parameters

`in`	*name*	Short error tag
`in`	*description*	Long error description

virtual crypto::customError::~customError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.8.3 Member Function Documentation

std::string crypto::customError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "<description>" (**crypto::customError::_description** (p. 79))

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::customError::errorTitle ( ) const `[inline]`, `[virtual]`

Short error descriptor Returns "<name>" (**crypto::customError::_name** (p. 79))

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.8.4 Member Data Documentation

std::string crypto::customError::_description `[private]`

@ Long error descriptor

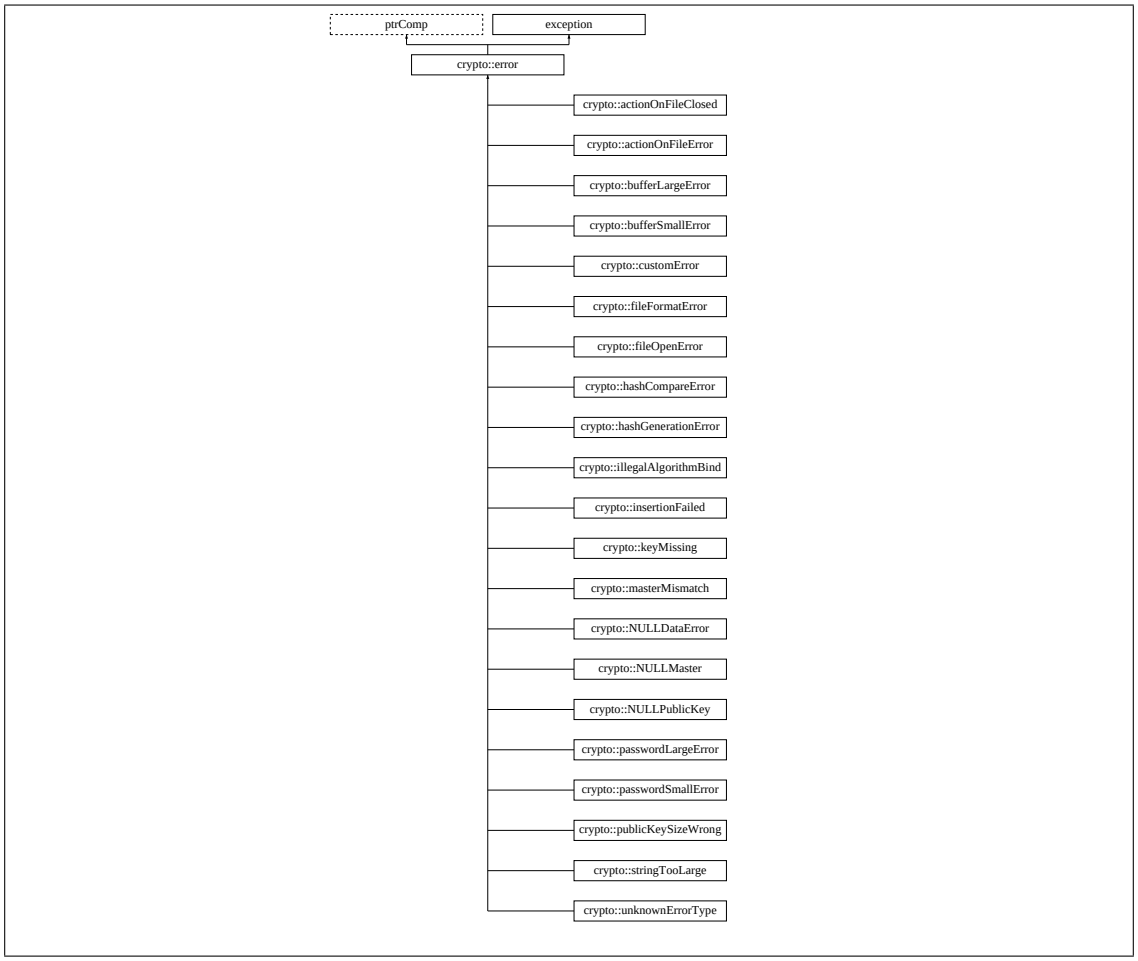
std::string crypto::customError::_name `[private]`

@ Short error descriptor

6.9 crypto::error Class Reference

Sortable exception.

Inheritance diagram for crypto::error:



Public Member Functions

- **error** ()

  Error constructor.
- virtual ~**error** () throw ()

  Virtual destructor.
- virtual std::string **errorTitle** () const

  Short error descriptor Returns "Error".
- virtual std::string **errorDescription** () const

  Long error descriptor Returns "No description".
- std::string **timestampString** () const

  Timestamp converted to string Returns the timestamp in a human readable string.
- void **log** () const

  Logs error to crypto::cryptoerr Logs the error title, time created and error description on the Crypto↩ Gateway error log.
- uint64_t **timestamp** () const

Time created.

- const char * **what** () const throw ()

  Concatenated error data Returns a C string of the error title, time constructed and error description.

Private Attributes

- uint64_t **_timestamp**

  Time the error was created.

- std::string **whatString**

  Full error output.

6.9.1 Detailed Description

Sortable exception.

This class allows for more sophisticated logging of errors. It contains the time which the error occurred and can be thrown.

6.9.2 Constructor & Destructor Documentation

crypto::error::error ( ) `[inline]`

Error constructor.

Constructs an error by setting the timestamp to the current time.

virtual crypto::error::~error ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.9.3 Member Function Documentation

virtual std::string crypto::error::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "No description".

Returns

Error description std::string

Reimplemented in **crypto::stringTooLarge** (p. 250), **crypto::unknownErrorType** (p. 251), **crypto::masterMismatch** (p. 146), **crypto::NULLMaster** (p. 172), **crypto::NULLDataError** (p. 171), **crypto::NULLPublicKey** (p. 173), **crypto::keyMissing** (p. 145), **crypto::publicKeySizeWrong** (p. 221), **crypto::actionOnFileClosed** (p. 56), **crypto::actionOnFileError** (p. 57), **crypto::hashGenerationError** (p. 122), **crypto::hashCompareError** (p. 120), **crypto::illegalAlgorithmBind** (p. 123), **crypto::fileFormatError** (p. 89), **crypto::fileOpenError** (p. 90), **crypto::customError** (p. 78), **crypto::insertionFailed** (p. 124), **crypto::bufferLargeError** (p. 76), **crypto::bufferSmallError** (p. 77), **crypto::passwordLargeError** (p. 193), and **crypto::passwordSmallError** (p. 195).

virtual std::string crypto::error::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Error".

Returns

Error title std::string

Reimplemented in **crypto::stringTooLarge** (p. 250), **crypto::unknownErrorType** (p. 252), **crypto::masterMismatch** (p. 147), **crypto::NULLMaster** (p. 172), **crypto::NULLDataError** (p. 171), **crypto::NULLPublicKey** (p. 173), **crypto::keyMissing** (p. 145), **crypto::publicKeySizeWrong** (p. 222), **crypto::actionOnFileClosed** (p. 56), **crypto::actionOnFileError** (p. 57), **crypto::hashGenerationError** (p. 122), **crypto::hashCompareError** (p. 121), **crypto::illegalAlgorithmBind** (p. 123), **crypto::fileFormatError** (p. 89), **crypto::fileOpenError** (p. 90), **crypto::customError** (p. 78), **crypto::insertionFailed** (p. 125), **crypto::bufferLargeError** (p. 76), **crypto::bufferSmallError** (p. 77), **crypto::passwordLargeError** (p. 194), and **crypto::passwordSmallError** (p. 195).

void crypto::error::log ( ) const [inline]

Logs error to crypto::cryptoerr Logs the error title, time created and error description on the CryptoGateway error log.

Returns

void

uint64_t crypto::error::timestamp ( ) const [inline]

Time created.

Returns

crypto::error::_timestamp (p. 82)

std::string crypto::error::timestampString ( ) const [inline]

Timestamp converted to string Returns the timestamp in a human readable string.

Returns

Time error was created

const char* crypto::error::what ( ) const throw) [inline]

Concatenated error data Returns a C string of the error title, time constructed and error description.

Returns

Character pointer to error data

6.9.4 Member Data Documentation

uint64_t crypto::error::_timestamp [private]

Time the error was created.

std::string crypto::error::whatString [private]

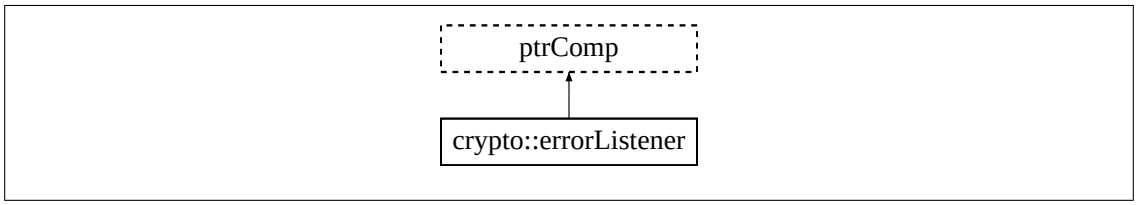
Full error output.

The **crypto::error::what()** (p. 82) function must return a C string. This string is the position in memory that function returns. **crypto::error::what()** (p. 82) also constructs this string.

6.10 crypto::errorListener Class Reference

crypto::error (p. 79) listener

Inheritance diagram for crypto::errorListener:



Public Member Functions

- virtual ~**errorListener** ()

  Virtual destructor.
- virtual void **receiveError** (**errorPointer** elm, os::smart_ptr< **errorSender** > source)

  Receive error event.

Private Attributes

- os::spinLock **mtx**

  Set protection mutex.
- os::smartSet< **errorSender** > **senders**

  Set of senders.

Friends

- class **errorSender**

  *Friendship with **crypto::errorSender** (p. 84).*

6.10.1 Detailed Description

crypto::error (p. 79) listener

Defines a class which is notified when another class throws a **crypto::error** (p. 79).

6.10.2 Constructor & Destructor Documentation

virtual crypto::errorListener::~errorListener ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.10.3 Member Function Documentation

virtual void crypto::errorListener::receiveError (**errorPointer** elm, os::smart_ptr< **errorSender** > source) [inline], [virtual]

Receive error event.

Receives error from one of the senders this listener is registered to.

Parameters

in	*elm*	Error sent
in	*source*	Sender which sent error

Returns

void

6.10.4 Friends And Related Function Documentation

friend class **errorSender** [friend]

Friendship with **crypto::errorSender** (p. 84).

The error sender must be able to add and remove itself from the listener's set.

6.10.5 Member Data Documentation

os::spinLock crypto::errorListener::mtx [private]

Set protection mutex.

Protects access to the set of senders, allows for multi-threading.

os::smartSet<**errorSender**> crypto::errorListener::senders [private]

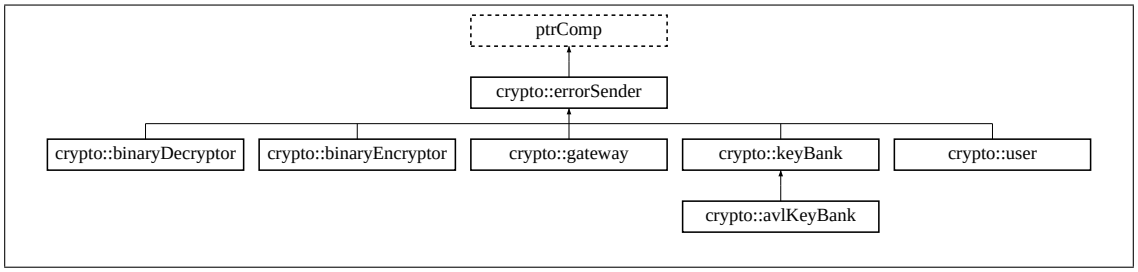
Set of senders.

All of the senders this listener is registered to.

6.11 crypto::errorSender Class Reference

Sends **crypto::error** (p. 79).

Inheritance diagram for crypto::errorSender:



Public Member Functions

- **errorSender** ()

  Error sender constructor.
- virtual ~**errorSender** ()

  Virtual destructor.
- void **pushErrorListener** (os::smart_ptr< **errorListener** > listener)

  Register listener.
- void **removeErrrorListener** (os::smart_ptr< **errorListener** > listener)

  Un-register listener.
- **errorPointer popError** ()

  Removes error from log.
- void **setLogLength** (unsigned int **logLength**)

  Set length of log.
- unsigned int **logLength** () const

  Return length of log.
- unsigned int **numberErrors** () const

  Return number of errors in log.

Protected Member Functions

- virtual void **logError** (**errorPointer** elm)

  Logs an error Dispatches an event to all listeners and stores the error in the log.

Private Attributes

- os::spinLock **listenerLock**

  Set protection mutex.
- os::smartSet< **errorListener** > **errorListen**

  Set of listeners.
- os::unsortedList< **error** > **errorLog**

  List of current errors.
- unsigned int **_logLength**

  Number of errors kept.

Friends

- class **errorListener**

  *Friendship with **crypto::errorListener** (p. 83).*

6.11.1 Detailed Description

Sends **crypto::error** (p. 79).

Sends and logs crypto:error pointers. Does not catch the errors, simply logs ones which have already been created and caught.

6.11.2 Constructor & Destructor Documentation

crypto::errorSender::errorSender ( ) `[inline]`

Error sender constructor.

Sets the length of the log to 20. Initializes with no errors and no listeners

virtual crypto::errorSender::~errorSender ( ) `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.11.3 Member Function Documentation

virtual void crypto::errorSender::logError (**errorPointer** elm) `[protected]`, `[virtual]`

Logs an error Dispatches an event to all listeners and stores the error in the log.

Parameters

`in`	*elm*	Error to be logged

Returns

void

Reimplemented in **crypto::gateway** (p. 98).

unsigned int crypto::errorSender::logLength ( ) const `[inline]`

Return length of log.

Returns

crypto::errorSender::_logLength (p. 88)

unsigned int crypto::errorSender::numberErrors ( ) const `[inline]`

Return number of errors in log.

Returns

crypto::errorSender::errorLog.size()

errorPointer crypto::errorSender::popError ( )

Removes error from log.

Returns

Oldest recorded error

void crypto::errorSender::pushErrorListener (os::smart_ptr< **errorListener** > listener)

Register listener.

Parameters

[in/out]	listener Listener to register

Returns

void

void crypto::errorSender::removeErrrorListener (os::smart_ptr< **errorListener** > listener)

Un-register listener.

Parameters

`in`	*listener*	Listener to un-register

Returns

void

void crypto::errorSender::setLogLength (unsigned int logLength)

Set length of log.

Parameters

`in`	*logLength*	Target length of log

Returns

void

6.11.4 Friends And Related Function Documentation

friend class **errorListener** [friend]

Friendship with **crypto::errorListener** (p. 83).
The error listener must be able to add and remove itself from the sender's set.

6.11.5 Member Data Documentation

unsigned int crypto::errorSender::_logLength [private]

Number of errors kept.
Allows for old errors to expire in the event a sender logs a lot of errors.

os::smartSet<**errorListener**> crypto::errorSender::errorListen [private]

Set of listeners.
All of the listeners registered to this sender.

os::unsortedList<**error**> crypto::errorSender::errorLog [private]

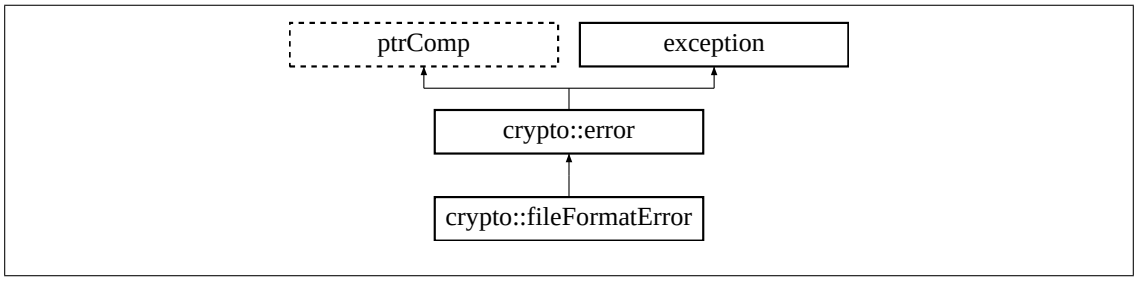
List of current errors.

os::spinLock crypto::errorSender::listenerLock [private]

Set protection mutex.
Protects access to the set of listeners, allows for multi-threading.

6.12 crypto::fileFormatError Class Reference

File format error.
Inheritance diagram for crypto::fileFormatError:



Public Member Functions

- virtual ~**fileFormatError** () throw ()
  Virtual destructor.
- std::string **errorTitle** () const
  Short error descriptor Returns "File Format Error".
- std::string **errorDescription** () const
  Long error descriptor Returns "The file is not of the specified format, and an error resulted".

6.12.1 Detailed Description

File format error.

Thrown when a file is parsed but an error occurs while parsing.

6.12.2 Constructor & Destructor Documentation

virtual crypto::fileFormatError::~fileFormatError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.12.3 Member Function Documentation

std::string crypto::fileFormatError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "The file is not of the specified format, and an error resulted".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::fileFormatError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "File Format Error".

Returns

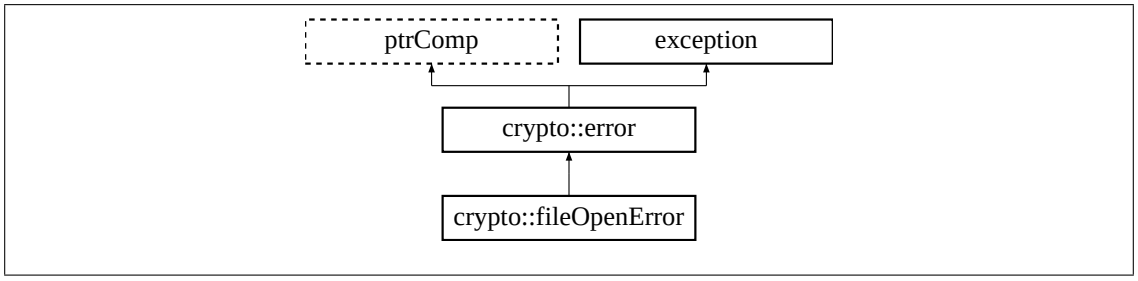
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.13 crypto::fileOpenError Class Reference

File open error.

Inheritance diagram for crypto::fileOpenError:



Public Member Functions

- virtual ~**fileOpenError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "File Open Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot open the specified file".

6.13.1 Detailed Description

File open error.

Thrown when a file cannot be found in the specified location.

6.13.2 Constructor & Destructor Documentation

virtual crypto::fileOpenError::~fileOpenError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.13.3 Member Function Documentation

std::string crypto::fileOpenError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot open the specified file".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::fileOpenError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "File Open Error".

Returns

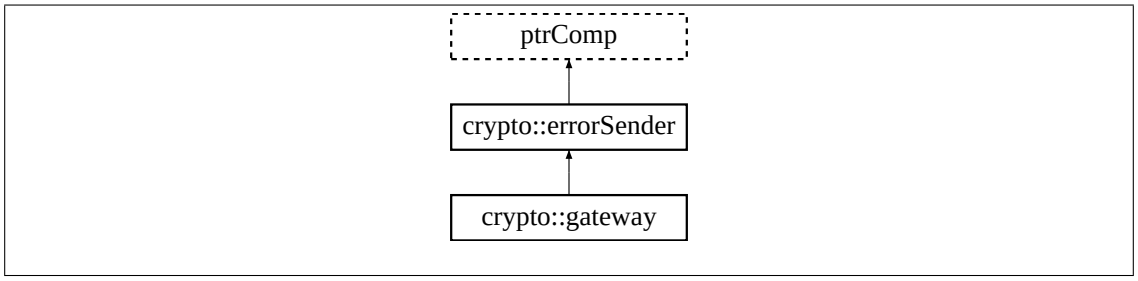
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.14 crypto::gateway Class Reference

Security gateway.

Inheritance diagram for crypto::gateway:



Public Member Functions

- **gateway** (os::smart_ptr< **user** > usr, std::string groupID="default")
  Gateway constructor.
- virtual ~**gateway** ()
  Virtual destructor.
- os::smart_ptr< **nodeGroup** > **brotherNode** ()
  Return the node group of the brother.
- os::smart_ptr< **message** > **getMessage** ()
  Returns next message from the gateway.
- os::smart_ptr< **message** > **send** (os::smart_ptr< **message** > msg)
  Send message through the gateway.
- os::smart_ptr< **message** > **ping** ()
  Ping message.
- os::smart_ptr< **message** > **processMessage** (os::smart_ptr< **message** > msg)
  Process incoming message.
- void **processTimestamps** ()
  Cycle time-stamp data.
- os::smart_ptr< **gatewaySettings** > **getBrotherSettings** ()
  Access brother settings.
- os::smart_ptr< **gatewaySettings** > **getSelfSettings** ()
  Access self settings.
- uint8_t **currentState** () const
  This gateway's status.
- uint8_t **brotherState** () const
  Brother gateway status.
- bool **secure** () const
  Gateway security established.
- uint64_t **timeout** () const
  Current receiver-side timeout value.
- uint64_t **safeTimeout** () const
  Current sender-side timeout value.
- uint64_t **errorTimeout** () const
  Current error timeout value.

- uint64_t **timeMessageReceived** () const

  Time-stamp of the last received message.
- uint64_t **timeMessageSent** () const

  Time-stamp of the last sent message.
- uint64_t **timeLastError** () const

  Time-stamp of the last error.

Static Public Attributes

- static const uint64_t **DEFAULT_TIMEOUT** =60

  Default timeout in seconds.
- static const uint64_t **DEFAULT_ERROR_TIMEOUT** =10

  Default error timeout in seconds.
- static const uint8_t **UNKNOWN_STATE** =0

  Unknown state value.
- static const uint8_t **UNKNOWN_BROTHER** =1

  Unknown brother state.
- static const uint8_t **SETTINGS_EXCHANGED** =2

  Settings exchanged state.
- static const uint8_t **ESTABLISHING_STREAM** =3

  Establishing stream state.
- static const uint8_t **STREAM_ESTABLISHED** =4

  Stream established state.
- static const uint8_t **SIGNING_STATE** =5

  Signing state.
- static const uint8_t **CONFIRM_OLD** =6

  Confirm old key state.
- static const uint8_t **ESTABLISHED** =7

  Stream established state.
- static const uint8_t **CONFIRM_ERROR_STATE** =252

  Confirm brother error state.
- static const uint8_t **BASIC_ERROR_STATE** =253

  Basic error state.
- static const uint8_t **TIMEOUT_ERROR_STATE** =254

  Timeout error state.
- static const uint8_t **PERMENANT_ERROR_STATE** =255

  Permanent error state.

Protected Member Functions

- void **logError** (**errorPointer** elm, uint8_t errType)

  Logs an error, with an error type.
- void **logError** (**errorPointer** elm)

  Logs an error, with type basic.

Private Member Functions

- void **clearStream** ()

  Resets stream tracking.
- void **buildStream** ()

  Builds the output stream.
- os::smart_ptr< **message** > **encrypt** (os::smart_ptr< **message** > msg)

  Encrypt a message.
- os::smart_ptr< **message** > **decrypt** (os::smart_ptr< **message** > msg)

  Decrypt a message.
- os::smart_ptr< **message** > **currentError** ()

  Build current error message.
- void **purgeLastError** ()

  Reset error.

Private Attributes

- os::smart_ptr< **gatewaySettings** > **selfSettings**

  Settings of this gateway.
- os::smart_ptr< **gatewaySettings** > **brotherSettings**

  Settings of the reciprocal gateway.
- os::spinLock **lock**

  Mutex protected gateway states.
- os::spinLock **stampLock**

  Mutex protecting timestamps.
- uint8_t **_currentState**

  Current state of this gateway.
- uint8_t **_brotherState**

  State of the reciprocal gateway.
- **errorPointer _lastError**

  Hold the most recent error.
- uint8_t **_lastErrorLevel**

  Holds the level of the last error.
- uint64_t **_errorTimestamp**

  Time-stamp of the last error.
- uint64_t **_timeout**

  Number of seconds till timeout.
- uint64_t **_safeTimeout**

  Number of seconds till partial timeout.
- uint64_t **_errorTimeout**

  Number of seconds for error timeout.
- uint64_t **_messageReceived**

  Time-stamp of last message received.
- uint64_t **_messageSent**

Time-stamp of last message sent.

- os::smart_ptr< **streamPackageFrame** > **selfStream**

  Stream algorithm for this gateway.
- os::smart_ptr< **publicKeyPackageFrame** > **selfPKFrame**

  Public key algorithm for this gateway.
- os::smart_ptr< **publicKey** > **selfPublicKey**

  Public/private key pair.
- os::smart_ptr< **number** > **selfPreciseKey**

  Public key for this gateway.
- os::smart_ptr< **streamPackageFrame** > **brotherStream**

  Stream algorithm for brother gateway.
- os::smart_ptr< **publicKeyPackageFrame** > **brotherPKFrame**

  Public key algorithm for bro.
- os::smart_ptr< **number** > **brotherPublicKey**

  Public key for brother gateway.
- os::smart_ptr< **message** > **streamMessageIn**

  Stream defining message: in.
- os::smart_ptr< **streamDecrypter** > **inputStream**

  Stream for incoming messages.
- uint64_t **streamEstTimestamp**

  Time the output stream was defined.
- os::smart_ptr< **message** > **streamMessageOut**

  Stream defining message: out.
- os::smart_ptr< **streamEncrypter** > **outputStream**

  Stream for outgoing messages.
- os::smart_ptr< uint8_t > **outputHashArray**

  Data for outgoing hashes.
- uint16_t **outputHashLength**

  Length of outgoing hash array.
- os::smart_ptr< **hash** > **selfPrimarySignatureHash**

  Hash for primary signature.
- os::smart_ptr< **hash** > **selfSecondarySignatureHash**

  Hash for historical signature.
- os::smart_ptr< **message** > **selfSigningMessage**

  Signing message: out.
- os::unsortedList< **hash** > **eligibleKeys**

  List of eligible public keys.
- os::smart_ptr< uint8_t > **inputHashArray**

  Data for incoming hashes.
- uint16_t **inputHashLength**

  Length of incoming hash array.
- os::smart_ptr< **hash** > **brotherPrimarySignatureHash**

  Hash of brother's primary signature.
- os::smart_ptr< **hash** > **brotherSecondarySignatureHash**

  Hash of brother's historical signature.

6.14.1 Detailed Description

Security gateway.

This gateway establishes a secured connection between two users. The connection uses the preferred algorithms as defined by the user.

6.14.2 Constructor & Destructor Documentation

crypto::gateway::gateway (os::smart_ptr< **user** > usr, std::string groupID = "default")

Gateway constructor.

Constructs a gateway from a user and a group ID. This initializes all gateway variables and binds the user settings to this gateway.

Parameters

in	*usr*	User sending information through this gateway
in	*groupID*	Defines group ID, "default" by default

virtual crypto::gateway::~gateway ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.14.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::gateway::brotherNode ( )

Return the node group of the brother.

Uses the current key bank to find the node associated with this brother.

Returns

brother node

uint8_t crypto::gateway::brotherState ( ) const [inline]

Brother gateway status.

Returns

gateway::_brotherState (p. 100)

void crypto::gateway::buildStream ( ) [private]

Builds the output stream.

Returns

void

void crypto::gateway::clearStream ( ) [private]

Resets stream tracking.

Resets all pointers defined while establishing a secure stream.

Returns

void

os::smart_ptr<**message**> crypto::gateway::currentError ( ) [private]

Build current error message.

Returns

Message

uint8_t crypto::gateway::currentState ( ) const [inline]

This gateway's status.

Returns

gateway::_currentState (p. 100)

os::smart_ptr<**message**> crypto::gateway::decrypt (os::smart_ptr< **message** > msg) [private]

Decrypt a message.

Uses the established input stream to decrypt the provided message and return it as a new message.

Parameters

in	*msg*	Message to be decrypted

Returns

Decrypted message

os::smart_ptr<**message**> crypto::gateway::encrypt (os::smart_ptr< **message** > msg) [private]

Encrypt a message.

Uses the established output stream to encrypt the provided message and return it as a new message.

Parameters

in	*msg*	Message to be encrypted

Returns

Encrypted message

uint64_t crypto::gateway::errorTimeout ( ) const [inline]

Current error timeout value.

Returns

gateway::_errorTimeout (p. 100)

os::smart_ptr<**gatewaySettings**> crypto::gateway::getBrotherSettings ( ) [inline]

Access brother settings.

Returns

Pointer to brother settings

os::smart_ptr<**message**> crypto::gateway::getMessage ( )

Returns next message from the gateway.

The function only returns the next message from the gateway's perspective. Gateway management messages are returned by this function.

Returns

Next management message

os::smart_ptr<**gatewaySettings**> crypto::gateway::getSelfSettings ( ) [inline]

Access self settings.

Returns

Pointer to self settings

void crypto::gateway::logError (**errorPointer** elm, uint8_t errType) [protected]

Logs an error, with an error type.

Wraps the "logError" funciton as defined by the **crypto::errorSender** (p. 84) class, also sets this particular gateway into some error state.

Parameters

in	*elm*	Error description
in	*errType*	Error level to determine timeout

Returns

void

void crypto::gateway::logError (**errorPointer** elm) [inline], [protected], [virtual]

Logs an error, with type basic.
Sets this particular gateway into a default error state by calling "logError" with a type.

Parameters

in	*elm*	Eror description

Returns

void

Reimplemented from **crypto::errorSender** (p. 86).

os::smart_ptr<**message**> crypto::gateway::ping ( )

Ping message.
Returns the ping message as defined by the **gatewaySettings** (p. 105) in this gateway.

Returns

Ping message for this user

os::smart_ptr<**message**> crypto::gateway::processMessage (os::smart_ptr< **message** > msg)

Process incoming message.
Decrypts and processes an incoming message. Note that messages must be coming from the brother gateway of this gateway.

Parameters

in	*msg*	Message to be processed

Returns

Decrypted message

void crypto::gateway::processTimestamps ( )

Cycle time-stamp data.
Compares registered time-stamps with the current time to determine if any state changes need to be made.

Returns

void

void crypto::gateway::purgeLastError ( ) [private]

Reset error.

Resets all error variables and returns the gateway to its unconnected state.

Returns

void

uint64_t crypto::gateway::safeTimeout ( ) const [inline]

Current sender-side timeout value.

Returns

gateway::_safeTimeout (p. 101)

bool crypto::gateway::secure ( ) const [inline]

Gateway security established.

Returns

true if established, else, false

os::smart_ptr<**message**> crypto::gateway::send (os::smart_ptr< **message** > msg)

Send message through the gateway.

Takes a message and encrypts it with the gateway, assuming the secure stream has been established. Returns an encrypted version of the message sent through the gateway.

Parameters

in	*msg*	Message to be encrypted

Returns

Encrypted message

uint64_t crypto::gateway::timeLastError ( ) const [inline]

Time-stamp of the last error.

Returns

gateway::_errorTimestamp (p. 100)

uint64_t crypto::gateway::timeMessageReceived ( ) const [inline]

Time-stamp of the last received message.

Returns

gateway::_messageReceived (p. 100)

uint64_t crypto::gateway::timeMessageSent ( ) const [inline]

Time-stamp of the last sent message.

Returns

gateway::_messageSent (p. 101)

uint64_t crypto::gateway::timeout ( ) const [inline]

Current receiver-side timeout value.

Returns

gateway::_timeout (p. 101)

6.14.4 Member Data Documentation

uint8_t crypto::gateway::_brotherState [private]

State of the reciprocal gateway.

uint8_t crypto::gateway::_currentState [private]

Current state of this gateway.

uint64_t crypto::gateway::_errorTimeout [private]

Number of seconds for error timeout.

When dealing with a timeout error, this defines how many seconds to wait before allowing a connection again.

uint64_t crypto::gateway::_errorTimestamp [private]

Time-stamp of the last error.

errorPointer crypto::gateway::_lastError [private]

Hold the most recent error.

This holds logging information for the most recent serious error. If an error is thrown while in an error state, the more serious error is kept in this variable.

uint8_t crypto::gateway::_lastErrorLevel [private]

Holds the level of the last error.

Either Basic, timeout or permanent. These are 253, 254 and 255 respectively.

uint64_t crypto::gateway::_messageReceived [private]

Time-stamp of last message received.

uint64_t crypto::gateway::_messageSent [private]

Time-stamp of last message sent.

uint64_t crypto::gateway::_safeTimeout [private]

Number of seconds till partial timeout.

This value is used as the timeout value when sending messages and is less than the timeout value so that receiving is more permissive than sending.

uint64_t crypto::gateway::_timeout [private]

Number of seconds till timeout.

This value is used when calculating timeout for receiving messages.

const uint8_t crypto::gateway::BASIC_ERROR_STATE =253 [static]

Basic error state.

A gateway has logged a low-level error. The connection must be re-set and re-established.

os::smart_ptr<**publicKeyPackageFrame**> crypto::gateway::brotherPKFrame [private]

Public key algorithm for bro.

os::smart_ptr<**hash**> crypto::gateway::brotherPrimarySignatureHash [private]

Hash of brother's primary signature.

If this hash is defined, then this gateway's brother has properly signed with the public key it declared.

os::smart_ptr<**number**> crypto::gateway::brotherPublicKey [private]

Public key for brother gateway.

os::smart_ptr<**hash**> crypto::gateway::brotherSecondarySignatureHash [private]

Hash of brother's historical signature.

When this hash is defined, this gateway's brother has properly signed with a historical public key.

os::smart_ptr<**gatewaySettings**> crypto::gateway::brotherSettings [private]

Settings of the reciprocal gateway.

Defined by the ping message which is received by this gateway's brother gateway.

os::smart_ptr<**streamPackageFrame**> crypto::gateway::brotherStream [private]

Stream algorithm for brother gateway.

const uint8_t crypto::gateway::CONFIRM_ERROR_STATE =252 [static]

Confirm brother error state.

In this state, a gateway is acknowledging to it's brother that the error notification sent by the brother was received and logged.

const uint8_t crypto::gateway::CONFIRM_OLD =6 [static]

Confirm old key state.

This indicates that a gateway has authenticated the identity of it's brother but has not been notified that its identity has been authenticated.

const uint64_t crypto::gateway::DEFAULT_ERROR_TIMEOUT =10 [static]

Default error timeout in seconds.

const uint64_t crypto::gateway::DEFAULT_TIMEOUT =60 [static]

Default timeout in seconds.

os::unsortedList<**hash**> crypto::gateway::eligibleKeys [private]

List of eligible public keys.

This list of hashes comes from the brother of this gateway. It is a list of the hashes of public keys associated with this node.

const uint8_t crypto::gateway::ESTABLISHED =7 [static]

Stream established state.

A secure and authentic stream has been established. Messages can be passed securely through the gateway.

const uint8_t crypto::gateway::ESTABLISHING_STREAM =3 [static]

Establishing stream state.

In this state, a gateway sends a symmetric stream key encrypted with the public key of the brother gateway.

os::smart_ptr<uint8_t> crypto::gateway::inputHashArray [private]

Data for incoming hashes.

uint16_t crypto::gateway::inputHashLength [private]

Length of incoming hash array.

os::smart_ptr<**streamDecrypter**> crypto::gateway::inputStream [private]

Stream for incoming messages.

os::spinLock crypto::gateway::lock [private]

Mutex protected gateway states.

os::smart_ptr<uint8_t> crypto::gateway::outputHashArray [private]

Data for outgoing hashes.

uint16_t crypto::gateway::outputHashLength [private]

Length of outgoing hash array.

os::smart_ptr<**streamEncrypter**> crypto::gateway::outputStream [private]

Stream for outgoing messages.

const uint8_t crypto::gateway::PERMENANT_ERROR_STATE =255 [static]

Permanent error state.

When gateways are in this state, a catastrophic error has occurred and the gateway refuses to reconnect.

os::smart_ptr<**publicKeyPackageFrame**> crypto::gateway::selfPKFrame [private]

Public key algorithm for this gateway.

os::smart_ptr<**number**> crypto::gateway::selfPreciseKey [private]

Public key for this gateway.

os::smart_ptr<**hash**> crypto::gateway::selfPrimarySignatureHash [private]

Hash for primary signature.

os::smart_ptr<**publicKey**> crypto::gateway::selfPublicKey [private]

Public/private key pair.

os::smart_ptr<**hash**> crypto::gateway::selfSecondarySignatureHash [private]

Hash for historical signature.

os::smart_ptr<**gatewaySettings**> crypto::gateway::selfSettings [private]

Settings of this gateway.

Defined by the user which constructed this gateway.

os::smart_ptr<**message**> crypto::gateway::selfSigningMessage [private]

Signing message: out.

This is a record of the message which was used to sign the current and historical public keys by this gateway in order to minimize the number of public key operations preformed.

os::smart_ptr<**streamPackageFrame**> crypto::gateway::selfStream [private]

Stream algorithm for this gateway.

const uint8_t crypto::gateway::SETTINGS_EXCHANGED =2 [static]

Settings exchanged state.

Indicates that a gateway has received a ping message from its reciprocal gateway, but has not received notification that the reciprocal gateway has received the ping message from this gateway.

const uint8_t crypto::gateway::SIGNING_STATE =5 [static]

Signing state.

Gateways in this state have established a secure stream with their brother node and now need to prove they have access to their declared public key. The signing message also contains hashes of keys associated with the particular node.

os::spinLock crypto::gateway::stampLock [private]

Mutex protecting timestamps.

const uint8_t crypto::gateway::STREAM_ESTABLISHED =4 [static]

Stream established state.

Gateways in this state continue to send the symmetric stream key, but also indicates to the brother gateway that the stream key sent by it has been received.

uint64_t crypto::gateway::streamEstTimestamp [private]

Time the output stream was defined.

Allows for redefinition of the output stream if the definition becomes stale.

os::smart_ptr<**message**> crypto::gateway::streamMessageIn [private]

Stream defining message: in.

This is a record of the message which defined the incoming stream in-order to minimize public key cryptography performed.

os::smart_ptr<**message**> crypto::gateway::streamMessageOut [private]

Stream defining message: out.

This is a record of the message which defined the outgoing stream in-order to minimize public key cryptography performed.

const uint8_t crypto::gateway::TIMEOUT_ERROR_STATE =254 [static]

Timeout error state.

Gateways are placed in this state when an error occurs while authenticating the connection. Because an error in this state is usually both expensive and indicative of unauthorized access, when errors occur, this state forces a certain amount of time in the error state before allowing reconnection.

const uint8_t crypto::gateway::UNKNOWN_BROTHER =1 [static]

Unknown brother state.

A gateway is in this state when it is unaware of the gateway settings of its reciprocal, or brother, gateway. In short, a gateway which does not know its brother has not received a ping.

const uint8_t crypto::gateway::UNKNOWN_STATE =0 [static]

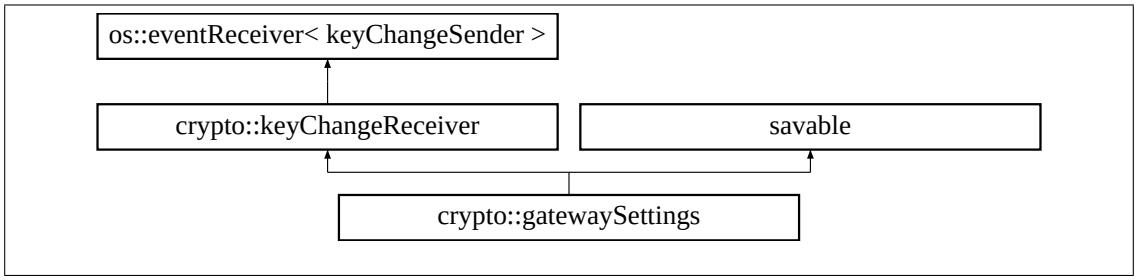
Unknown state value.

This state is used by a gateway when the it is not aware of the current state of its reciprocal gateway. A gateway should never be in this state itself.

6.15 crypto::gatewaySettings Class Reference

Holds settings for gateway encryption.

Inheritance diagram for crypto::gatewaySettings:



Public Member Functions

- **gatewaySettings** (os::smart_ptr< **user** > usr, std::string **groupID**, std::string **filePath**="")

  User constructor.
- **gatewaySettings** (const **message** &msg)

  Ping message constructor.
- virtual ~**gatewaySettings** ()

  Virtual destructor.
- os::smartXMLNode **generateSaveTree** ()

  Generate XML save stream.
- void **update** ()

  Ensure preferred algorithms are defined.
- void **save** ()

Saves the class to a file Saves the settings to an XML file, if the file path is defined.

- void **load** ()

  Loads the class from a file Loads the settings from an XML file, if the file path is defined.
- const std::string & **filePath** () const

  Return reference to the file path.
- const std::string & **groupID** () const

  Return reference to the group ID.
- const std::string & **nodeName** () const

  Return reference to the node name.
- os::smart_ptr< **user** > **getUser** ()

  Return user, if it is defined.
- os::smart_ptr< **publicKey** > **getPrivateKey** ()

  Return public/private key pair, if it is defined.
- os::smart_ptr< **number** > **getPublicKey** ()

  Return public key.
- uint16_t **prefferedPublicKeyAlgo** () const

  Return public key algorithm ID.
- uint16_t **prefferedPublicKeySize** () const

  Return public key algorithm size.
- uint16_t **prefferedHashAlgo** () const

  Return hash algorithm ID.
- uint16_t **prefferedHashSize** () const

  Return hash size.
- uint16_t **prefferedStreamAlgo** () const

  Return stream algorithm ID.
- os::smart_ptr< **message** > **ping** ()

  Construct a ping message.
- bool **operator==** (const **gatewaySettings** &cmp) const

  Equality comparison operator.
- bool **operator!=** (const **gatewaySettings** &cmp) const

  Not-equals comparison operator.
- bool **operator**< (const **gatewaySettings** &cmp) const

  Less-than comparison operator.
- bool **operator**> (const **gatewaySettings** &cmp) const

  Greater-than comparison operator.
- bool **operator**<**=** (const **gatewaySettings** &cmp) const

  Less-than/Equals-to comparison operator.
- bool **operator**>**=** (const **gatewaySettings** &cmp) const

  Greater-than/Equals-to comparison operator.

Public Attributes

- os::multiLock **lock**

  Read/write mutex.

Protected Member Functions

- void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggered when the public key is changed.

Private Attributes

- std::string **_groupID**

  Group ID of the node, unique to this settings class.
- std::string **_nodeName**

  Name of the node, defined by the user.
- std::string **_filePath**

  Save file path.
- os::smart_ptr< **user** > **_user**

  Pointer to the user class.
- os::smart_ptr< **publicKey** > **_privateKey**

  Pointer to public/private key pair.
- os::smart_ptr< **number** > **_publicKey**

  Pointer to the public key.
- uint16_t **_prefferedPublicKeyAlgo**

  Public key algorithm ID.
- uint16_t **_prefferedPublicKeySize**

  Public key size (uint32_t size)
- uint16_t **_prefferedHashAlgo**

  Hash algorithm ID.
- uint16_t **_prefferedHashSize**

  Hash size (in bytes)
- uint16_t **_prefferedStreamAlgo**

  Stream algorithm ID.

6.15.1 Detailed Description

Holds settings for gateway encryption.

Contains all of the information needed to define how the gateway functions. This includes which algorithms are white-listed, which are black- listed and which are preferred. Note that this settings class can define the settings for a node whose private key is known or for a node whose private key is unknown.

6.15.2 Constructor & Destructor Documentation

crypto::gatewaySettings::gatewaySettings (os::smart_ptr< **user** > usr, std::string groupID, std::string filePath = "")

User constructor.

Constructs the class from a user. While this constructor can be called outside the user class, it is suggested to use the interface provided in **crypto::user** (p. 252) to create new gateway settings.

Parameters

in	*usr*	User defining the settings
in	*groupID*	Group ID of the settings
in	*filePath*	Save file location (optional)

crypto::gatewaySettings::gatewaySettings (const **message** & msg)

Ping message constructor.

Constructs the gateway settings from a ping message. This is usually used by the gateway to parse ping messages it receives.

Parameters

in	*msg*	Ping message

virtual crypto::gatewaySettings::~gatewaySettings ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.15.3 Member Function Documentation

const std::string& crypto::gatewaySettings::filePath ( ) const [inline]

Return reference to the file path.

Returns

gatewaySettings::_filePath (p. 112)

os::smartXMLNode crypto::gatewaySettings::generateSaveTree ( )

Generate XML save stream.

Returns

XML save tree

os::smart_ptr<**publicKey**> crypto::gatewaySettings::getPrivateKey ( ) [inline]

Return public/private key pair, if it is defined.

Returns

gatewaySettings::_privateKey (p. 113)

os::smart_ptr<**number**> crypto::gatewaySettings::getPublicKey ( ) `[inline]`

Return public key.

Returns

gatewaySettings::_publicKey (p. 113)

os::smart_ptr<**user**> crypto::gatewaySettings::getUser ( ) `[inline]`

Return user, if it is defined.

Returns

gatewaySettings::_user (p. 113)

const std::string& crypto::gatewaySettings::groupID ( ) const `[inline]`

Return reference to the group ID.

Returns

gatewaySettings::_groupID (p. 113)

void crypto::gatewaySettings::load ( )

Loads the class from a file Loads the settings from an XML file, if the file path is defined.

Returns

void

const std::string& crypto::gatewaySettings::nodeName ( ) const `[inline]`

Return reference to the node name.

Returns

gatewaySettings::_nodeName (p. 113)

bool crypto::gatewaySettings::operator!= (const **gatewaySettings** & cmp) const `[inline]`

Not-equals comparison operator.
Uses the group ID to gateway settings.

Parameters

`in`	*cmp*	Object to compare against

Returns

this->_groupID != cmp._groupID

bool crypto::gatewaySettings::operator< (const **gatewaySettings** & cmp) const [inline]

Less-than comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID < cmp._groupID

bool crypto::gatewaySettings::operator<= (const **gatewaySettings** & cmp) const [inline]

Less-than/Equals-to comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID <= cmp._groupID

bool crypto::gatewaySettings::operator== (const **gatewaySettings** & cmp) const [inline]

Equality comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID == cmp._groupID

bool crypto::gatewaySettings::operator> (const **gatewaySettings** & cmp) const [inline]

Greater-than comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID > cmp._groupID

bool crypto::gatewaySettings::operator>= (const **gatewaySettings** & cmp) const [inline]

Greater-than/Equals-to comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID >= cmp._groupID

os::smart_ptr<**message**> crypto::gatewaySettings::ping ( )

Construct a ping message.

Returns

New ping message

uint16_t crypto::gatewaySettings::prefferedHashAlgo ( ) const [inline]

Return hash algorithm ID.

Returns

gatewaySettings::_prefferedHashAlgo (p. 113)

uint16_t crypto::gatewaySettings::prefferedHashSize ( ) const [inline]

Return hash size.

Returns

gatewaySettings::_prefferedHashSize (p. 113)

uint16_t crypto::gatewaySettings::prefferedPublicKeyAlgo ( ) const [inline]

Return public key algorithm ID.

Returns

gatewaySettings::_prefferedPublicKeyAlgo (p. 113)

uint16_t crypto::gatewaySettings::prefferedPublicKeySize ( ) const [inline]

Return public key algorithm size.

Returns

gatewaySettings::_prefferedPublicKeySize (p. 113)

uint16_t crypto::gatewaySettings::prefferedStreamAlgo ( ) const [inline]

Return stream algorithm ID.

Returns

gatewaySettings::_prefferedStreamAlgo (p. 113)

void crypto::gatewaySettings::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) [protected], [virtual]

Triggered when the public key is changed.

Updates the gateway settings when the user indicates a public key has been updated.

Parameters

in	*pbk*	Updated public/private key pair

Returns

void

Reimplemented from **crypto::keyChangeReceiver** (p. 143).

void crypto::gatewaySettings::save ( )

Saves the class to a file Saves the settings to an XML file, if the file path is defined.

Returns

void

void crypto::gatewaySettings::update ( )

Ensure preferred algorithms are defined.

Uses current information in the class to determine if known algorithms define the preferred algorithms in this class. If the preferred algorithms are not defined, they are changed to defined algorithms.

Returns

void

6.15.4 Member Data Documentation

std::string crypto::gatewaySettings::_filePath [private]

Save file path.

If the setting was defined by the user and not a "ping" message, it will often have a save file location.

std::string crypto::gatewaySettings::_groupID [private]

Group ID of the node, unique to this settings class.

std::string crypto::gatewaySettings::_nodeName [private]

Name of the node, defined by the user.

uint16_t crypto::gatewaySettings::_prefferedHashAlgo [private]

Hash algorithm ID.

uint16_t crypto::gatewaySettings::_prefferedHashSize [private]

Hash size (in bytes)

uint16_t crypto::gatewaySettings::_prefferedPublicKeyAlgo [private]

Public key algorithm ID.

uint16_t crypto::gatewaySettings::_prefferedPublicKeySize [private]

Public key size (uint32_t size)

uint16_t crypto::gatewaySettings::_prefferedStreamAlgo [private]

Stream algorithm ID.

os::smart_ptr<**publicKey**> crypto::gatewaySettings::_privateKey [private]

Pointer to public/private key pair.

os::smart_ptr<**number**> crypto::gatewaySettings::_publicKey [private]

Pointer to the public key.

os::smart_ptr<**user**> crypto::gatewaySettings::_user [private]

Pointer to the user class.

os::multiLock crypto::gatewaySettings::lock

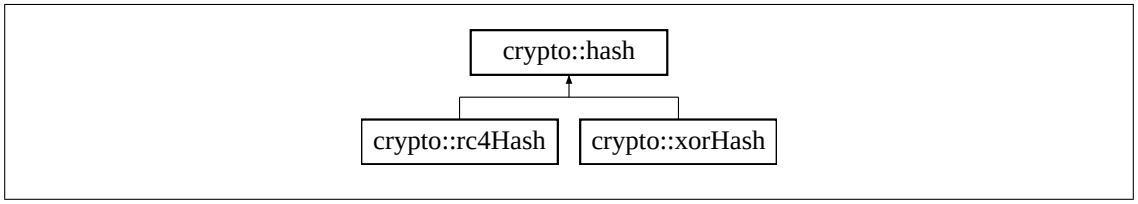
Read/write mutex.

When this class is defined by a user, it is possible for the user to change the gateway settings during runtime. Because of this, a read/write lock is required.

6.16 crypto::hash Class Reference

Base hash class.

Inheritance diagram for crypto::hash:



Public Member Functions

- **hash** (const **hash** &cpy)

  Hash copy constructor.
- **hash** & **operator=** (const **hash** &cpy)

  Equality constructor.
- virtual ~**hash** ()

  Virtual destructor.
- int **compare** (const **hash** *_comp) const

  Comparison function.
- virtual void **preformHash** (unsigned char ***data**, uint32_t dLen)

  Binds a data-set.
- virtual std::string **algorithmName** () const

  Algorithm name string access.
- uint16_t **algorithm** () const

  Current algorithm ID.
- uint16_t **size** () const

  Current hash size.
- uint32_t **numBits** () const

  Current hash size, bits.
- unsigned char * **data** ()

  Modifiable data access.
- const unsigned char * **data** () const

  Constant data access.
- unsigned char **operator[ ]** (uint16_t pos) const

  Modifiable data access.
- unsigned char & **operator[ ]** (uint16_t pos)

  Constant data access.
- std::string **toString** () const

  Converts hash to string.
- void **fromString** (const std::string &str)

  Converts from string.

- bool **operator==** (const **hash** &comp) const
- bool **operator!=** (const **hash** &comp) const
- bool **operator**> (const **hash** &comp) const
- bool **operator**>**=** (const **hash** &comp) const
- bool **operator**< (const **hash** &comp) const
- bool **operator**<**=** (const **hash** &comp) const

Static Public Member Functions

- static std::string **staticAlgorithmName** ()

  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()

  Algorithm ID number access.

Protected Member Functions

- **hash** (uint16_t **algorithm**=algo::hashNULL, uint16_t **size**=size::defaultHash)

  Default hash constructor.

Protected Attributes

- uint16_t **_size**

  Number of bytes in the hash.
- unsigned char * **_data**

  Raw hash data.

Private Attributes

- uint16_t **_algorithm**

  Hash algorithm ID.

6.16.1 Detailed Description

Base hash class.

This class manages the raw data of all hashes. Subsequent hashes define different algorithms to populate the hashes.

6.16.2 Constructor & Destructor Documentation

crypto::hash::hash (uint16_t algorithm = `algo::hashNULL`, uint16_t size = `size::defaultHash`) `[protected]`

Default hash constructor.

Constructs a hash with the given size and algorithm ID, initializing the entire hash itself to 0.

Parameters

in	*algorithm*	Algorithm ID, NULL by default
in	*size*	Size of hash, crypto::size::defaultHash by default

crypto::hash::hash (const **hash** & cpy)

Hash copy constructor.

Constructs a hash with a hash. This copy constructor re-initializes the data array for the new hash.

Parameters

in	*cpy*	Hash to copy

virtual crypto::hash::~hash ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.16.3 Member Function Documentation

uint16_t crypto::hash::algorithm ( ) const [inline]

Current algorithm ID.

Returns the algorithm ID bound to this hash.

Returns

crypto::hash::_algorithm (p. 119)

virtual std::string crypto::hash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.

Returns the name of the current algorithm string. This function is virtual, so changes for each hash algorithm

Returns

"NULL"

Reimplemented in **crypto::xorHash** (p. 263), and **crypto::rc4Hash** (p. 236).

int crypto::hash::compare (const **hash** * _comp) const

Comparison function.

Takes into consideration the algorithm, size of the data and content of the hash. Used for all of the equality operators.

Returns

0 if equal, 1 if greater than, -1 if less than

unsigned char* crypto::hash::data ( ) [inline]

Modifiable data access.
Provides mutable data-access to the raw hash data.

Returns

crypto::hash::_data (p. 119)

const unsigned char* crypto::hash::data ( ) const [inline]

Constant data access.
Provides immutable data-access to the raw hash data.

Returns

crypto::hash::_data (p. 119)

void crypto::hash::fromString (const std::string & str)

Converts from string.
Rebuilds the hash from a hex string.

Parameters

in	*str*	Hex string

Returns

String representation of the hash

uint32_t crypto::hash::numBits ( ) const [inline]

Current hash size, bits.
Return the hash size bound to this hash in bits.

Returns

crypto::hash::_size (p. 119)*8

bool crypto::hash::operator!= (const **hash** & comp) const [inline]

bool crypto::hash::operator< (const **hash** & comp) const [inline]

bool crypto::hash::operator<= (const **hash** & comp) const [inline]

hash& crypto::hash::operator= (const **hash** & cpy)

Equality constructor.
Rebuild this hash with the data from another hash.

Parameters

in	*cpy*	Hash to copy

Returns

Reference to this

bool crypto::hash::operator== (const **hash** & comp) const [inline]

bool crypto::hash::operator> (const **hash** & comp) const [inline]

bool crypto::hash::operator>= (const **hash** & comp) const [inline]

unsigned char crypto::hash::operator[ ] (uint16_t pos) const

Modifiable data access.

Provides mutable data-access to the raw hash data.

Parameters

in	*pos*	Data index

Returns

crypto::hash::_data (p. 119)[pos]

unsigned char& crypto::hash::operator[ ] (uint16_t pos)

Constant data access.

Provides immutable data-access to the raw hash data.

Parameters

in	*pos*	Data index

Returns

crypto::hash::_data (p. 119)[pos]

virtual void crypto::hash::preformHash (unsigned char * data, uint32_t dLen) [inline], [virtual]

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

uint16_t crypto::hash::size ( ) const [inline]

Current hash size.
Returns the hash size bound to this hash in bytes.

Returns

crypto::hash::_size (p. 119)

static uint16_t crypto::hash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.
Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashNULL

static std::string crypto::hash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.
Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"NULL"

std::string crypto::hash::toString ( ) const

Converts hash to string.
Converts the hash to a hex string.

Returns

String representation of the hash

6.16.4 Member Data Documentation

uint16_t crypto::hash::_algorithm [private]

Hash algorithm ID.

unsigned char* crypto::hash::_data [protected]

Raw hash data.

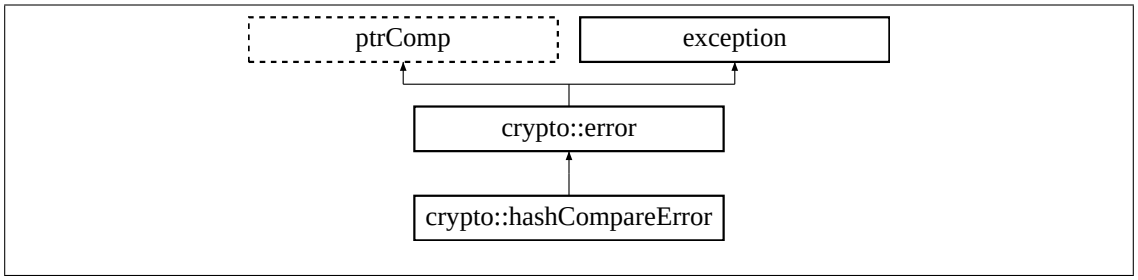
uint16_t crypto::hash::_size [protected]

Number of bytes in the hash.

6.17 crypto::hashCompareError Class Reference

Hash mis-match.

Inheritance diagram for crypto::hashCompareError:



Public Member Functions

- virtual ~**hashCompareError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Hash Compare".
- std::string **errorDescription** () const

  Long error descriptor Returns "Provided and calculated hashes do not match".

6.17.1 Detailed Description

Hash mis-match.

Thrown when two hashes do not match. This error can be indicative of larger security issues, as it most commonly occurs during a failed authentication.

6.17.2 Constructor & Destructor Documentation

virtual crypto::hashCompareError::~hashCompareError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.17.3 Member Function Documentation

std::string crypto::hashCompareError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Provided and calculated hashes do not match".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::hashCompareError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Hash Compare".

Returns

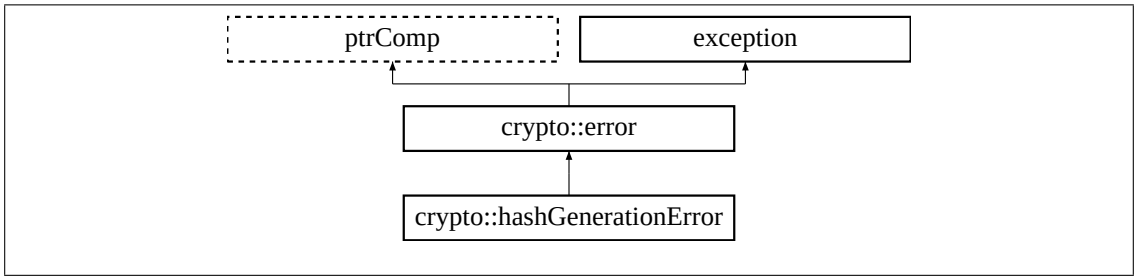
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.18 crypto::hashGenerationError Class Reference

Hash generation error.

Inheritance diagram for crypto::hashGenerationError:



Public Member Functions

- virtual ~**hashGenerationError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Hash Generation".
- std::string **errorDescription** () const

  Long error descriptor Returns "Could not generate a hash with the given arguments".

6.18.1 Detailed Description

Hash generation error.

Thrown when a hash encounters an error while being created.

6.18.2 Constructor & Destructor Documentation

virtual crypto::hashGenerationError::~hashGenerationError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.18.3 Member Function Documentation

std::string crypto::hashGenerationError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Could not generate a hash with the given arguments".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::hashGenerationError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Hash Generation".

Returns

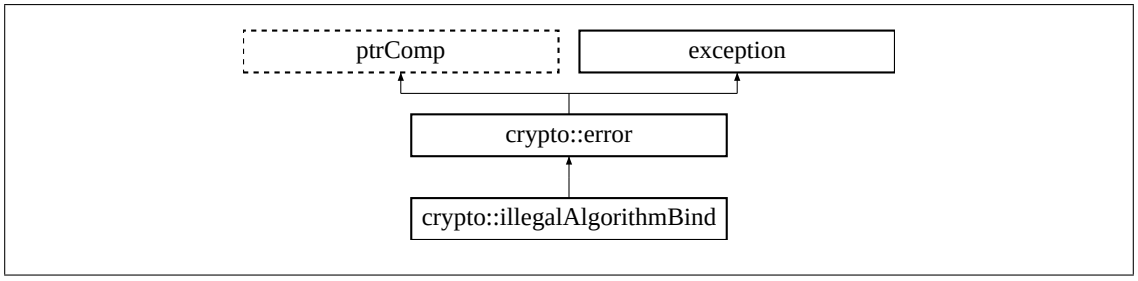
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.19 crypto::illegalAlgorithmBind Class Reference

Algorithm bound failure.

Inheritance diagram for crypto::illegalAlgorithmBind:



Public Member Functions

- **illegalAlgorithmBind** (std::string algoName)

  Illegal algorithm error.
- virtual ~**illegalAlgorithmBind** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Illegal Algorithm Bind".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot bind algorithm of type: <algorithmName>".

Private Attributes

- std::string **algorithmName**

  Name of algorithm.

6.19.1 Detailed Description

Algorithm bound failure.

Thrown when an algorithm cannot be found or used. Usually indicates the specified algorithm is not defined by the active version.

6.19.2 Constructor & Destructor Documentation

crypto::illegalAlgorithmBind::illegalAlgorithmBind (std::string algoName) `[inline]`

Illegal algorithm error.

Parameters

`in`	*algoName*	Name of illegal algorithm

virtual crypto::illegalAlgorithmBind::~illegalAlgorithmBind ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.19.3 Member Function Documentation

std::string crypto::illegalAlgorithmBind::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Cannot bind algorithm of type: <algorithmName>".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::illegalAlgorithmBind::errorTitle ( ) const `[inline]`, `[virtual]`

Short error descriptor Returns "Illegal Algorithm Bind".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.19.4 Member Data Documentation

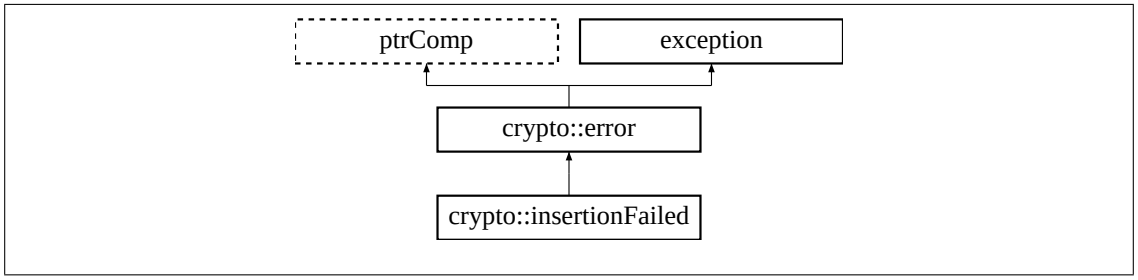
std::string crypto::illegalAlgorithmBind::algorithmName [private]

Name of algorithm.

6.20 crypto::insertionFailed Class Reference

ADS Insertion Failed.

Inheritance diagram for crypto::insertionFailed:



Public Member Functions

- virtual ~**insertionFailed** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Insertion Failed".
- std::string **errorDescription** () const

  Long error descriptor Returns "Insertion into an abstract data-structure unexpectedly failed".

6.20.1 Detailed Description

ADS Insertion Failed.

Thrown when insertion to an os::ads structure unexpectedly fails.

6.20.2 Constructor & Destructor Documentation

virtual crypto::insertionFailed::~insertionFailed ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.20.3 Member Function Documentation

std::string crypto::insertionFailed::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Insertion into an abstract data-structure unexpectedly failed".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::insertionFailed::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Insertion Failed".

Returns

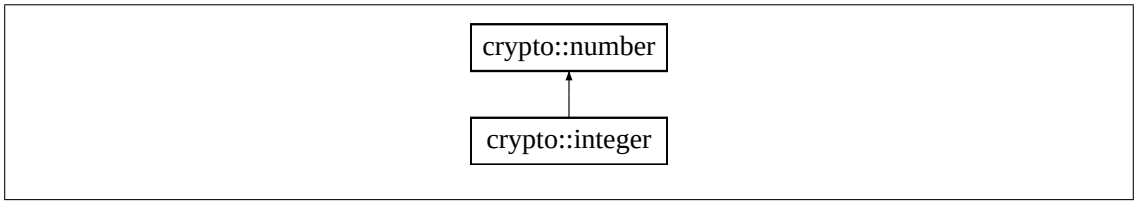
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.21 crypto::integer Class Reference

Integer number definition.

Inheritance diagram for crypto::integer:



Public Member Functions

- **integer** ()
  Default integer constructor.
- **integer** (uint16_t **size**)
  Construct integer with size.
- **integer** (const uint32_t *d, uint16_t **size**)
  Construct integer with data array.
- **integer** (const **integer** &num)
  Copy constructor.
- virtual ~**integer** ()
  Virtual destructor.
- bool **checkType** () const
  Check if the number is valid.
- **operator number** () const
  Allows integer to be cast as a number.
- **integer operator+** (const **integer** &n) const
  Integer addition operator.
- **integer** & **operator+=** (const **integer** &n)

Integer addition equals operator.

- **integer** & **operator++** ()

  Increment operator.
- **integer operator++** (int dummy)

  Increment operator.
- **integer operator-** (const **integer** &n) const

  Integer subtraction operator.
- **integer** & **operator-=** (const **integer** &n)

  Integer subtraction equals operator.
- **integer** & **operator--** ()

  Decrement operator.
- **integer operator--** (int dummy)

  Decrement operator.
- **integer operator**>> (uint16_t n) const

  Right shift operator.
- **integer operator**<< (uint16_t n) const

  Left shift operator.
- **integer operator**∗ (const **integer** &n) const

  Integer multiplication operator.
- **integer** & **operator**∗**=** (const **integer** &n)

  Integer multiplication equals operator.
- **integer operator**/ (const **integer** &n) const

  Integer division operator.
- **integer** & **operator/=** (const **integer** &n)

  Integer division equals operator.
- **integer operator%** (const **integer** &n) const

  Integer modulo operator.
- **integer** & **operator%=** (const **integer** &n)

  Integer modulo equals operator.
- **integer exponentiation** (const **integer** &n) const

  Integer exponentiation function.
- **integer** & **exponentiationEquals** (const **integer** &n)

  Integer exponentiation equals function.
- **integer moduloExponentiation** (const **integer** &n, const **integer** &mod) const

  Integer modulo-exponentiation function.
- **integer** & **moduloExponentiationEquals** (const **integer** &n, const **integer** &mod)

  Integer modulo-exponentiation equals function.
- **integer gcd** (const **integer** &n) const

  Integer GCD function.
- **integer** & **gcdEquals** (const **integer** &n)

  Integer GCD equals function.
- **integer modInverse** (const **integer** &m) const

  Integer modular inverse function.

- **integer** & **modInverseEquals** (const **integer** &n)

  Integer modular inverse equals function.
- bool **prime** (uint16_t testVal=algo::primeTestCycle) const

  Test if this integer is prime.

Static Public Member Functions

- static **integer zero** ()

  Constructs a '0' integer.
- static **integer one** ()

  Constructs a '1' integer.
- static **integer two** ()

  Constructs a '2' integer.

Additional Inherited Members

6.21.1 Detailed Description

Integer number definition.
A traditional numerical definition which can be of arbitrary size.

6.21.2 Constructor & Destructor Documentation

crypto::integer::integer ( )

Default integer constructor.

crypto::integer::integer (uint16_t size)

Construct integer with size.

Parameters

in	*size*	Size integer is initialized with

crypto::integer::integer (const uint32_t * d, uint16_t size)

Construct integer with data array.

Parameters

in	*d*	Data array to be bound
in	*size*	Size of array

crypto::integer::integer (const **integer** & num)

Copy constructor.

Parameters

in	*num*	Integer used to construct this

virtual crypto::integer::~integer ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.21.3 Member Function Documentation

bool crypto::integer::checkType ( ) const [virtual]

Check if the number is valid.

Checks to ensure that the number definition for this object is the Base-10 type. Ensure that all basic mathematical operators are defined.

Returns

true if valid type, else, false

Reimplemented from **crypto::number** (p. 179).

integer crypto::integer::exponentiation (const **integer** & n) const

Integer exponentiation function.

Parameters

in	*n*	Integer to be raised to

Returns

this^n

integer& crypto::integer::exponentiationEquals (const **integer** & n)

Integer exponentiation equals function.

Parameters

in	*n*	Integer to be raised to

Returns

this = this^n

integer crypto::integer::gcd (const **integer** & n) const

Integer GCD function.

Parameters

in	*n*	Integer to be compared against

Returns

GCD of this and n

integer& crypto::integer::gcdEquals (const **integer** & n)

Integer GCD equals function.

Parameters

in	*n*	Integer to be compared against

Returns

this = GCD of this and n

integer crypto::integer::modInverse (const **integer** & m) const

Integer modular inverse function.

Parameters

in	*n*	Integer representing modulo space

Returns

(this^-1) % n

integer& crypto::integer::modInverseEquals (const **integer** & n)

Integer modular inverse equals function.

Parameters

in	*n*	Integer representing modulo space

Returns

this = (this^-1) % n

integer crypto::integer::moduloExponentiation (const **integer** & n, const **integer** & mod) const

Integer modulo-exponentiation function.

Parameters

in	*n*	Integer to be raised to
in	*mod*	Integer representing modulo space

Returns

this^n % mod

integer& crypto::integer::moduloExponentiationEquals (const **integer** & n, const **integer** & mod)

Integer modulo-exponentiation equals function.

Parameters

in	*n*	Integer to be raised to
in	*mod*	Integer representing modulo space

Returns

this = this^n % mod

static **integer** crypto::integer::one ( ) [static]

Constructs a '1' integer.

Returns

1

crypto::integer::operator **number** ( ) const [inline]

Allows integer to be cast as a number.

Returns

number(*this)

integer crypto::integer::operator% (const **integer** & n) const

Integer modulo operator.

Parameters

in	*n*	Integer defining modulo space this % n

integer& crypto::integer::operator%= (const **integer** & n)

Integer modulo equals operator.

Parameters

in	*n*	Integer defining modulo space this = this % n

integer crypto::integer::operator∗ (const **integer** & n) const

Integer multiplication operator.

Parameters

in	*n*	Integer to be multiplied this ∗ n

integer& crypto::integer::operator∗= (const **integer** & n)

Integer multiplication equals operator.

Parameters

in	*n*	Integer to be multiplied this = this ∗ n

integer crypto::integer::operator+ (const **integer** & n) const

Integer addition operator.

Parameters

in	*n*	Integer to be added this + n

integer& crypto::integer::operator++ ( )

Increment operator.

Returns

this++

integer crypto::integer::operator++ (int dummy)

Increment operator.

Returns

++this

integer& crypto::integer::operator+= (const **integer** & n)

Integer addition equals operator.

Parameters

in	*n*	Integer to be added this = this + n

integer crypto::integer::operator- (const **integer** & n) const

Integer subtraction operator.

Parameters

in	*n*	Integer to be subtracted this - n

integer& crypto::integer::operator-- ( )

Decrement operator.

Returns

this–

integer crypto::integer::operator-- (int dummy)

Decrement operator.

Returns

–this

integer& crypto::integer::operator-= (const **integer** & n)

Integer subtraction equals operator.

Parameters

in	*n*	Integer to be subtracted this = this - n

integer crypto::integer::operator/ (const **integer** & n) const

Integer division operator.

Parameters

in	*n*	Integer to be divided by this / n

integer& crypto::integer::operator/= (const **integer** & n)

Integer division equals operator.

Parameters

in	*n*	Integer to be divided by this = this / n

integer crypto::integer::operator<< (uint16_t n) const

Left shift operator.

Parameters

in	*n*	Number of bits to shift

Returns

this << n

integer crypto::integer::operator>> (uint16_t n) const

Right shift operator.

Parameters

in	*n*	Number of bits to shift

Returns

this >> n

bool crypto::integer::prime (uint16_t testVal = `algo::primeTestCycle`) const

Test if this integer is prime.

Preforms a probabilistic prime test on this number. This operation can be quite expensive, especially for large numbers.

Parameters

`in`	*testVal*	Number of test cycles, crytpo::algo::primeTestCycle by default

Returns

true if prime, else, false

static **integer** crypto::integer::two ( ) `[static]`

Constructs a '2' integer.

Returns

2

static **integer** crypto::integer::zero ( ) `[inline]`, `[static]`

Constructs a '0' integer.

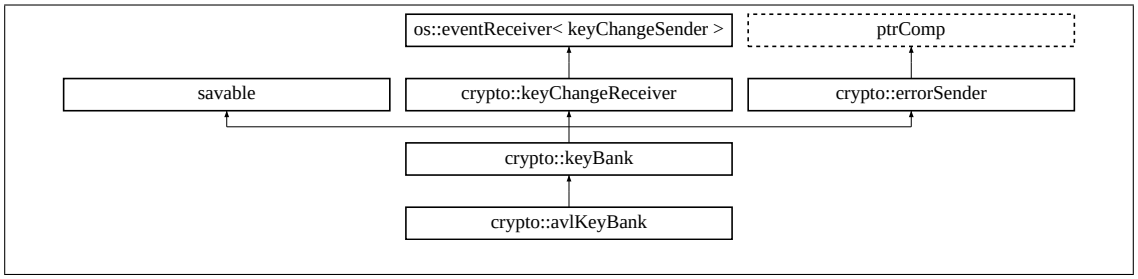
Returns

0

6.22 crypto::keyBank Class Reference

Key bank interface.

Inheritance diagram for crypto::keyBank:



Public Member Functions

- virtual ~**keyBank** ()

  Virtual destructor.

- virtual os::smart_ptr< **nodeGroup** > **addPair** (std::string groupName, std::string name, os↩ ::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)=0

  Adds authenticated node to bank.
- virtual void **save** ()=0

  Saves bank to file.
- const std::string & **savePath** () const

  Get save path.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeNameReference** > name)=0

  Find by group name reference.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeKeyReference** > key)=0

  Find by group key reference.
- virtual os::smart_ptr< **nodeGroup** > **find** (std::string groupName, std::string name)

  Find by group name and name.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Find by key information.
- void **setPassword** (const unsigned char ∗key=NULL, unsigned int keyLen=0)

  Set password.
- void **setStreamPackage** (os::smart_ptr< **streamPackageFrame** > strmPack)

  Set stream package.
- void **setPublicKey** (os::smart_ptr< **publicKey** > pubKey)

  Set public key.

Protected Member Functions

- virtual void **pushNewNode** (os::smart_ptr< **nodeNameReference** > name)=0

  Add name node.
- virtual void **pushNewNode** (os::smart_ptr< **nodeKeyReference** > key)=0

  Add key node.
- virtual void **load** ()=0

  Loads bank from file.
- void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggers on key change.
- os::smart_ptr< **nodeGroup** > **fileLoadHelper** (os::smartXMLNode xmlTree)

  Construct node with XML tree.
- **keyBank** (std::string **savePath**, const unsigned char ∗key=NULL, unsigned int keyLen=0, os↩ ::smart_ptr< **streamPackageFrame** > strmPck=NULL)

  Construct with save path and key.
- **keyBank** (std::string **savePath**, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **stream**↩ **PackageFrame** > strmPck=NULL)

  Construct with save path and public key.

Protected Attributes

- os::smart_ptr< **streamPackageFrame** > **_streamPackage**

  Stream package.
- unsigned char * **_symKey**

  Primary symmetric key.
- unsigned int **_keyLen**

  Length of symmetric key.
- os::smart_ptr< **publicKey** > **_pubKey**

  Public key group to encrypt file.

Private Attributes

- std::string **_savePath**

  Path to save file.

Friends

- class **nodeGroup**

  Friendship with node grouping.

6.22.1 Detailed Description

Key bank interface.

Acts as an interface for classes which allow for the storing, saving and searching of cyptographic keys. These banks act, in essense, as data-bases.

6.22.2 Constructor & Destructor Documentation

crypto::keyBank::keyBank (std::string savePath, const unsigned char * key = `NULL`, unsigned int keyLen = `0`, os::smart_ptr< **streamPackageFrame** > strmPck = `NULL`) `[protected]`

Construct with save path and key.

Parameters

`in`	*savePath*	Path to save file
`in`	*key*	Symetric key
`in`	*keyLen*	Length of symetric key
`in`	*strmPck*	Definition of algorithms used

crypto::keyBank::keyBank (std::string savePath, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **streamPackageFrame** > strmPck = `NULL`) `[protected]`

Construct with save path and public key.

Parameters

in	*savePath*	Path to save file
in	*pubKey*	Public key
in	*strmPck*	Definition of algorithms used

virtual crypto::keyBank::~keyBank ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.22.3 Member Function Documentation

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::addPair (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [pure virtual]

Adds authenticated node to bank.

Note that if a node has not be authenticated, adding it to the bank will cause a potential security vulnerability. Nodes should be authenticated before being added to the bank.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node
in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Return reference to the new node group

Implemented in **crypto::avlKeyBank** (p. 59).

os::smart_ptr<**nodeGroup**> crypto::keyBank::fileLoadHelper (os::smartXMLNode xmlTree) [inline], [protected]

Construct node with XML tree.

Parameters

in	*xmlTree*	XML tree from file

Returns

Node group constructed with tree

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **nodeNameReference** > name) [pure virtual]

Find by group name reference.

Parameters

in	*name*	Name reference to be searched

Returns

Node group found by arguments

Implemented in **crypto::avlKeyBank** (p. 60).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **nodeKeyReference** > key) [pure virtual]

Find by group key reference.

Parameters

in	*key*	Key reference to be searched

Returns

Node group found by arguments

Implemented in **crypto::avlKeyBank** (p. 60).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (std::string groupName, std::string name) [inline], [virtual]

Find by group name and name.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node

Returns

Node group found by arguments

Reimplemented in **crypto::avlKeyBank** (p. 60).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [inline], [virtual]

Find by key information.

Parameters

in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Node group found by arguments

Reimplemented in **crypto::avlKeyBank** (p. 61).

virtual void crypto::keyBank::load ( ) [protected], [pure virtual]

Loads bank from file.

Returns

void

Implemented in **crypto::avlKeyBank** (p. 61).

void crypto::keyBank::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) [protected], [virtual]

Triggers on key change.
Marks this class for re-saving when the public key has been re-generated.

Parameters

in	*pbk*	Public key which was changed

Returns

void

Reimplemented from **crypto::keyChangeReceiver** (p. 143).

virtual void crypto::keyBank::pushNewNode (os::smart_ptr< **nodeNameReference** > name) [protected], [pure virtual]

Add name node.
Inserts a name node into the bank. The name node has a reference to a node group.

Parameters

in	*name*	Name node to be added

Returns

void

Implemented in **crypto::avlKeyBank** (p. 61).

virtual void crypto::keyBank::pushNewNode (os::smart_ptr< **nodeKeyReference** > key) [protected], [pure virtual]

Add key node.

Inserts a key node into the bank. The key node has a reference to a node group.

Parameters

in	*key*	Key node to be added

Returns

void

Implemented in **crypto::avlKeyBank** (p. 62).

virtual void crypto::keyBank::save ( ) [pure virtual]

Saves bank to file.

Returns

void

Implemented in **crypto::avlKeyBank** (p. 62).

const std::string& crypto::keyBank::savePath ( ) const [inline]

Get save path.

Returns

crypto::keyBank::_savePath (p. 141)

void crypto::keyBank::setPassword (const unsigned char * key = NULL, unsigned int keyLen = 0)

Set password.

Sets symetric key used to securely save user data.

Parameters

in	*key*	Symetric key
in	*keyLen*	Length of symetric key

Returns

void

void crypto::keyBank::setPublicKey (os::smart_ptr< **publicKey** > pubKey)

Set public key.
Binds a new public key to this. Calls for saving of this user.

Parameters

in	*pubKey*	Public key

Returns

void

void crypto::keyBank::setStreamPackage (os::smart_ptr< **streamPackageFrame** > strmPack)

Set stream package.
Binds a new stream package. Calls for saving of this user.

Parameters

in	*strmPack*	Stream package

Returns

void

6.22.4 Friends And Related Function Documentation

friend class **nodeGroup** [friend]

Friendship with node grouping.
Node groups must be able to push name and key nodes onto the key bank.

6.22.5 Member Data Documentation

unsigned int crypto::keyBank::_keyLen [protected]

Length of symmetric key.

os::smart_ptr<**publicKey**> crypto::keyBank::_pubKey [protected]

Public key group to encrypt file.

std::string crypto::keyBank::_savePath [private]

Path to save file.

os::smart_ptr<**streamPackageFrame**> crypto::keyBank::_streamPackage [protected]

Stream package.

Used for the saving of the key bank. This defines the algorithms used for encrypting the saved bank, if it is encrypted.

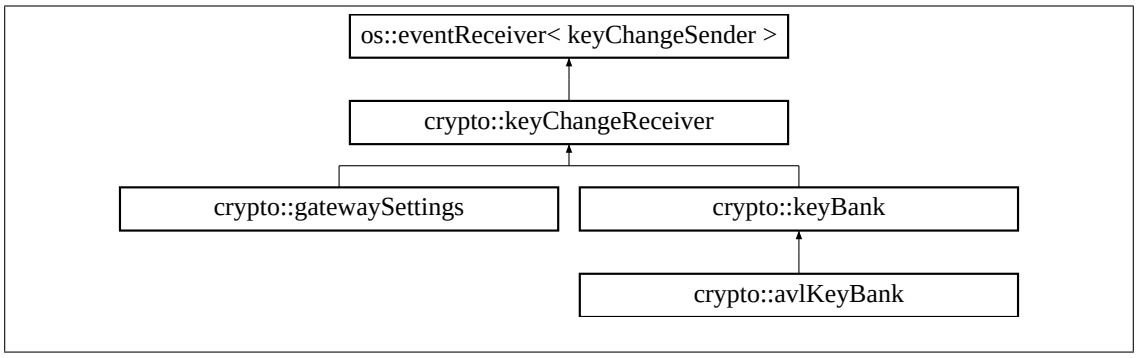
unsigned char* crypto::keyBank::_symKey [protected]

Primary symmetric key.

6.23 crypto::keyChangeReceiver Class Reference

Interface for receiving key changes.

Inheritance diagram for crypto::keyChangeReceiver:



Public Member Functions

- virtual ~**keyChangeReceiver** ()

  Virtual destructor.

Protected Member Functions

- virtual void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggers on key change.

Friends

- class **keyChangeSender**

  *Allows access to **crypto::keyChangeReceiver::publicKeyChanged** (p. 143).*

6.23.1 Detailed Description

Interface for receiving key changes.

A class which is alerted by public keys when the public key is updated.

6.23.2 Constructor & Destructor Documentation

virtual crypto::keyChangeReceiver::~keyChangeReceiver ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.23.3 Member Function Documentation

virtual void crypto::keyChangeReceiver::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) `[inline]`, `[protected]`, `[virtual]`

Triggers on key change.

Is triggered by **crypto::publicKey** (p. 195) whenever the public key is updated.

Parameters

`in`	*pbk*	Public key which was changed

Returns

void

Reimplemented in **crypto::keyBank** (p. 139), and **crypto::gatewaySettings** (p. 112).

6.23.4 Friends And Related Function Documentation

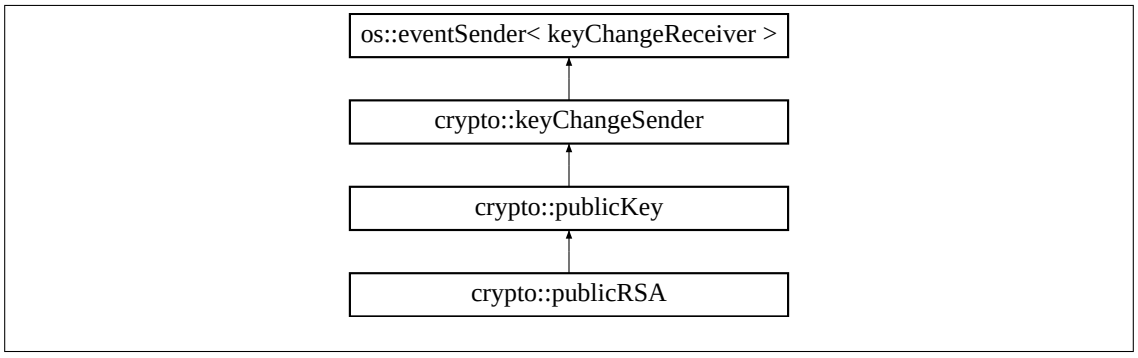
friend class **keyChangeSender** `[friend]`

Allows access to **crypto::keyChangeReceiver::publicKeyChanged** (p. 143).

6.24 crypto::keyChangeSender Class Reference

Interface inherited by **publicKey** (p. 195).

Inheritance diagram for crypto::keyChangeSender:



Public Member Functions

- virtual ~**keyChangeSender** ()

  Virtual destructor.

Protected Member Functions

- void **sendEvent** (os::smart_ptr< **keyChangeReceiver** > ptr)

  Sends key change event to listeners.

6.24.1 Detailed Description

Interface inherited by **publicKey** (p. 195).

This class is meaningless outside of **crypto::publicKey** (p. 195) and is only designed to be inherited by **publicKey** (p. 195) to interface with **crypto::keyChangeReceiver** (p. 142).

6.24.2 Constructor & Destructor Documentation

virtual crypto::keyChangeSender::~keyChangeSender ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.24.3 Member Function Documentation

void crypto::keyChangeSender::sendEvent (os::smart_ptr< **keyChangeReceiver** > ptr) `[inline]`, `[protected]`

Sends key change event to listeners.

Useing the interface provided by the os::eventSender class, alert any classes listening for a public key change that one has occured.

Parameters

`in`	*ptr*	Receiver to alert

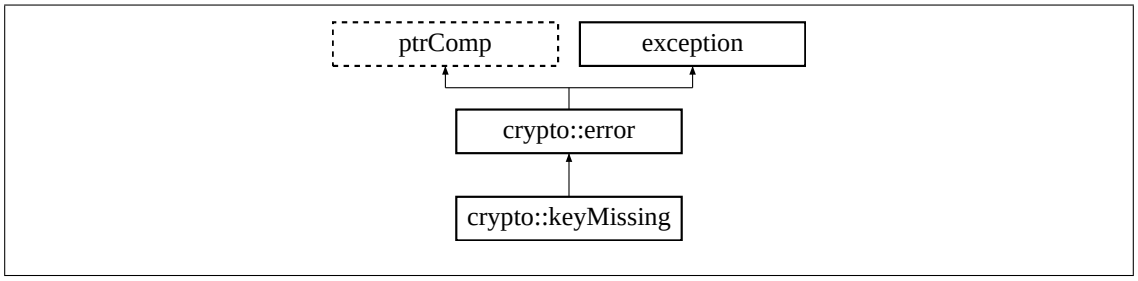
Returns

void

6.25 crypto::keyMissing Class Reference

Key missing error.

Inheritance diagram for crypto::keyMissing:



Public Member Functions

- virtual ~**keyMissing** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Key missing".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot decrypt the data stream, the key is missing!".

6.25.1 Detailed Description

Key missing error.

Thrown when a key cannot be found to decrypt the incoming data stream

6.25.2 Constructor & Destructor Documentation

virtual crypto::keyMissing::~keyMissing ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.25.3 Member Function Documentation

std::string crypto::keyMissing::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Cannot decrypt the data stream, the key is missing!".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::keyMissing::errorTitle ( ) const `[inline]`, `[virtual]`

Short error descriptor Returns "Key missing".

Returns

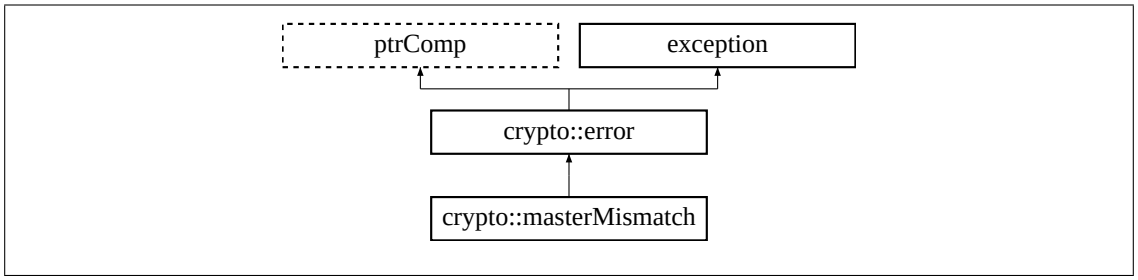
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.26 crypto::masterMismatch Class Reference

Master mis-match.

Inheritance diagram for crypto::masterMismatch:



Public Member Functions

- virtual ~**masterMismatch** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Master Comparison Mis-match".
- std::string **errorDescription** () const

  Long error descriptor Returns "Two nodes which are interacting have different masters!".

6.26.1 Detailed Description

Master mis-match.

Thrown when two elements attempt an interaction but have different masters.

6.26.2 Constructor & Destructor Documentation

virtual crypto::masterMismatch::~masterMismatch ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.26.3 Member Function Documentation

std::string crypto::masterMismatch::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Two nodes which are interacting have different masters!".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::masterMismatch::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Master Comparison Mis-match".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.27 crypto::message Class Reference

Crypto-Gateway message.

Public Member Functions

- **message** (uint16_t sz)
  Constructs message with a size.
- **message** (const **message** &msg)
  Copy constructor.
- virtual ~**message** ()
  Virtual destructor.
- uint16_t **messageSize** () const
  Return message size.
- uint16_t **size** () const
  Return message packet size.
- uint16_t **encryptionDepth** () const
  Return level of message encryption.
- uint8_t * **data** ()
  Modifiable data pointer.
- const uint8_t * **data** () const
  Immutable data pointer.
- bool **encrypted** () const
  Is the message encrypted.
- bool **pushString** (std::string s)
  Add string to this message.
- std::string **popString** ()
  Remove string from this message.

Static Public Member Functions

- static **message encryptedMessage** (uint8_t *rawData, uint16_t sz)
  Constructs an encrypted message.
- static **message decryptedMessage** (uint8_t *rawData, uint16_t sz)
  Constructs an decrypted message.

Static Public Attributes

- static const uint8_t **BLOCKED** =0
  Blocked message tag.
- static const uint8_t **PING** =1
  Ping message tag.
- static const uint8_t **FORWARD** =2
  Forward message tag.
- static const uint8_t **STREAM_KEY** =3
  Stream key message tag.
- static const uint8_t **SIGNING_MESSAGE** =4
  Signing message tag.
- static const uint8_t **SECURE_DATA_EXCHANGE** =5
  Secure data exchange message tag.
- static const uint8_t **CONFIRM_ERROR** =252
  Confirm error message tag.
- static const uint8_t **BASIC_ERROR** =253
  Basic error message tag.
- static const uint8_t **TIMEOUT_ERROR** =254
  Timeout error message tag.
- static const uint8_t **PERMENANT_ERROR** =255
  Permenant error message tag.

Private Attributes

- uint16_t **_messageSize**
  Size of message.
- uint16_t **_size**
  Size of the message packet.
- uint16_t **_encryptionDepth**
  Depth of encryption.
- uint8_t * **_data**
  Data in the message packet.

Friends

- class **gatewaySettings**
  Friendship with settings.
- class **gateway**
  Friendship with gateway.

6.27.1 Detailed Description

Crypto-Gateway message.

This message is meant to be passed between machines. The gateway either encrypts or decrypts the message. This message allows for nested encryption.

6.27.2 Constructor & Destructor Documentation

crypto::message::message (uint16_t sz)

Constructs message with a size.

Parameters

in	*sz*	Size of message

crypto::message::message (const **message** & msg)

Copy constructor.

Parameters

in	*msg*	Message to be copied

virtual crypto::message::~message ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.27.3 Member Function Documentation

uint8_t* crypto::message::data ( ) [inline]

Modifiable data pointer.

Returns

message::_data (p. 151)

const uint8_t* crypto::message::data ( ) const [inline]

Immutable data pointer.

Returns

message::_data (p. 151)

static **message** crypto::message::decryptedMessage (uint8_t * rawData, uint16_t sz) [static]

Constructs an decrypted message.

Parses an array of data assuming that the data in question has been generated outside of a gateway

Parameters

in	*rawData*	Incoming data array
in	*sz*	Size of incoming data

Returns

New message

bool crypto::message::encrypted ( ) const [inline]

Is the message encrypted.

Returns

True if encrypted, else, false

static **message** crypto::message::encryptedMessage (uint8_t * rawData, uint16_t sz) [static]

Constructs an encrypted message.
Parses an array of data assuming that the data in question has come out of another gateway.

Parameters

in	*rawData*	Incoming data array
in	*sz*	Size of incoming data

Returns

New message

uint16_t crypto::message::encryptionDepth ( ) const [inline]

Return level of message encryption.

Returns

message::_encryptionDepth (p. 151)

uint16_t crypto::message::messageSize ( ) const [inline]

Return message size.

Returns

message::_messageSize (p. 151)

std::string crypto::message::popString ( )

Remove string from this message.

Returns

Next string to remove

bool crypto::message::pushString (std::string s)

Add string to this message.

Returns

True if successful

uint16_t crypto::message::size ( ) const [inline]

Return message packet size.

Returns

message::_size (p. 152)

6.27.4 Friends And Related Function Documentation

friend class **gateway** [friend]

Friendship with gateway.

The **crypto::gateway** (p. 90) class encrypts and decrypts messages, so it must be able to access the intrisics of the message.

friend class **gatewaySettings** [friend]

Friendship with settings.

The **crypto::gatewaySettings** (p. 105) class must be able to access the intrinsics of this class inorder to create and parse a ping message.

6.27.5 Member Data Documentation

uint8_t* crypto::message::_data [private]

Data in the message packet.

uint16_t crypto::message::_encryptionDepth [private]

Depth of encryption.

Holds how many times this particular message has been encrypted.

uint16_t crypto::message::_messageSize [private]

Size of message.

This size refers to the size of the non-header and non-checksum bytes in the message. This value remains constant as messages are encrypted and decrypted.

uint16_t crypto::message::_size [private]

Size of the message packet.
This size includes all support data along with the meaningful message.

const uint8_t crypto::message::BASIC_ERROR =253 [static]

Basic error message tag.
Sent by a gateway when a basic error occurs.

const uint8_t crypto::message::BLOCKED =0 [static]

Blocked message tag.
Indicates that the node sending the particular message has blocked the node receiving the particular message.

const uint8_t crypto::message::CONFIRM_ERROR =252 [static]

Confirm error message tag.
Messages of this type are sent to allow the receiving gateway to know that the sending gateway has acknowledged its error.

const uint8_t crypto::message::FORWARD =2 [static]

Forward message tag.
Indicates a message is being sent through this gateway to another gateway for final decryption.

const uint8_t crypto::message::PERMENANT_ERROR =255 [static]

Permenant error message tag.
Sent by a gateway when a permenant error has occurred. Permenant errors never expire, and a gateway will never reconnect once a permenant error has occurred.

const uint8_t crypto::message::PING =1 [static]

Ping message tag.
Message type sent by gateways when exchanging names and public keys.

const uint8_t crypto::message::SECURE_DATA_EXCHANGE =5 [static]

Secure data exchange message tag.
Message passed between two gateways when secure. Used by the gateways to notify connected gateways when keys and algorithms change after a connection has been secured.

const uint8_t crypto::message::SIGNING_MESSAGE =4 [static]

Signing message tag.
Indicates a message is cryptographically establishing the identity of a node.

const uint8_t crypto::message::STREAM_KEY =3 [static]

Stream key message tag.

Indicates a message is exchanging stream cipher keys through the defined public key algorithm.

const uint8_t crypto::message::TIMEOUT_ERROR =254 [static]

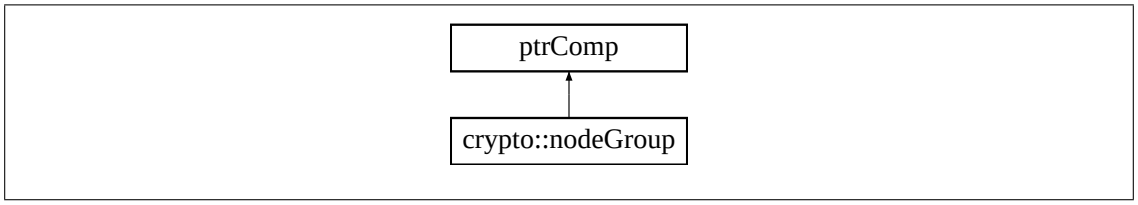
Timeout error message tag.

Sent by a gateway when a timeout error occurs. Timeout errors are more serious and take a certain amount of time to expire.

6.28 crypto::nodeGroup Class Reference

Node group.

Inheritance diagram for crypto::nodeGroup:



Public Member Functions

- **nodeGroup** (**keyBank** *master, std::string groupName, std::string **name**, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Node group constructor.
- virtual ~**nodeGroup** ()

  Virtual destructor.
- void **getName** (std::string &groupName, std::string &**name**)

  Allows access to the most recent name.
- std::string **name** ()

  Concatenated name.
- os::smart_ptr< os::adnode< **nodeNameReference** > > **getFirstName** ()

  Returns first name in the list.
- os::smart_ptr< os::adnode< **nodeKeyReference** > > **getFirstKey** ()

  Returns first key in the list.
- void **merge** (**nodeGroup** &source)

  Merge a node group into this.
- void **addAlias** (std::string groupName, std::string **name**, uint64_t timestamp=os::getTimestamp())

  Add new alias for group.
- void **addKey** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp=os↩ ::getTimestamp())

  Add new key for group.

- unsigned int **numberOfNames** () const

  Returns the number of names.
- unsigned int **numberOfKeys** () const

  Returns the number of keys.
- os::smart_ptr< os::smart_ptr< **nodeNameReference** > > **namesByTimestamp** (unsigned int &size)

  Returns names sorted by timestamp.
- os::smart_ptr< os::smart_ptr< **nodeKeyReference** > > **keysByTimestamp** (unsigned int &size)

  Returns keys sorted by timestamp.
- os::smartXMLNode **buildXML** ()

  Build XML tree.

Private Member Functions

- void **sortKeys** ()

  Sorts keys by timestamp.
- void **sortNames** ()

  Sorts names by timestamp.
- **nodeGroup** (**keyBank** *master, os::smartXMLNode fileNode)

  Node group constructor.

Private Attributes

- **keyBank** * **_master**

  Pointer to key bank.
- os::asyncAVLTree< **nodeNameReference** > **nameList**

  List of all names associated with this node.
- os::asyncAVLTree< **nodeKeyReference** > **keyList**

  List of all keys associated with this node.
- std::mutex **sortingLock**

  Lock used for sorting.
- os::smart_ptr< os::smart_ptr< **nodeNameReference** > > **sortedNames**

  Array of names sorted by timestamp.
- os::smart_ptr< os::smart_ptr< **nodeKeyReference** > > **sortedKeys**

  Array of keys sorted by timestamp.

Friends

- class **keyBank**

  *Only **keyBank** (p. 134) can load a node group.*

6.28.1 Detailed Description

Node group.

A list of all names and keys which are associated with a single node. This must exist because nodes can change their name during operation.

6.28.2 Constructor & Destructor Documentation

crypto::nodeGroup::nodeGroup (**keyBank** * master, os::smartXMLNode fileNode) `[private]`

Node group constructor.

Constructs a node group with an XML tree. This constructor throws exceptions if errors occur.

Parameters

	[in/out]	master Reference to the 'master' group holder
`in`	*fileNode*	XML root which defines the group

crypto::nodeGroup::nodeGroup (**keyBank** * master, std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

Node group constructor.

Parameters

	[in/out]	master Reference to the 'master' group holder
`in`	*groupName*	Group name of the node being registered
`in`	*name*	Name of the node being registered
`in`	*key*	The public key of a given node
`in`	*algoID*	The algorithm identifier
`in`	*keySize*	Size of the key provided

virtual crypto::nodeGroup::~nodeGroup ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.28.3 Member Function Documentation

void crypto::nodeGroup::addAlias (std::string groupName, std::string name, uint64_t timestamp = `os::getTimestamp()`)

Add new alias for group.

Parameters

`in`	*groupName*	Group name of the node being registered
`in`	*name*	Name of the node being registered
	timestamp	The time this node was created, 'now' by defult

Returns

void

void crypto::nodeGroup::addKey (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp = `os::getTimestamp()`)

Add new key for group.

Parameters

`in`	*key*	The public key of a given node
`in`	*algoID*	The algorithm identifier
`in`	*keySize*	Size of the key provided
	timestamp	The time this node was created, 'now' by defult

Returns

void

os::smartXMLNode crypto::nodeGroup::buildXML ( )

Build XML tree.

Builds an XML tree from this node group. This tree is designed to be saved by the key bank.

Returns

Root of tree to be saved

os::smart_ptr<os::adnode<**nodeKeyReference**> > crypto::nodeGroup::getFirstKey ( ) `[inline]`

Returns first key in the list.

This function returns an alphabetical order. Note that it is often the case that a user needs to sort by timestamp. This functionality is also provided.

Returns

crypto::nodeGroup::keyList.getFirst()

os::smart_ptr<os::adnode<**nodeNameReference**> > crypto::nodeGroup::getFirstName ( ) `[inline]`

Returns first name in the list.

This function returns an alphabetical order. Note that it is often the case that a user needs to sort by timestamp. This functionality is also provided.

Returns

crypto::nodeGroup::nameList.getFirst()

void crypto::nodeGroup::getName (std::string & groupName, std::string & name)

Allows access to the most recent name.

Parameters

out	*groupName*	**crypto::nodeGroup::sortedNames** (p. 159)[0]->groupName()
out	*name*	**crypto::nodeGroup::sortedNames** (p. 159)[0]->**name()** (p. 157)

Returns

void

os::smart_ptr<os::smart_ptr<**nodeKeyReference**> > crypto::nodeGroup::keysByTimestamp (unsigned int & size)

Returns keys sorted by timestamp.

Parameters

out	*size*	Size of array to be returned

Returns

crypto::nodeGroup::sortedKeys (p. 159)

void crypto::nodeGroup::merge (**nodeGroup** & source)

Merge a node group into this.

Acheives merge entirely by reference. It is assumed that the node being merged into this node will shortly be deleted.

Parameters

in	*source*	Node group to merge

Returns

void

std::string crypto::nodeGroup::name ( )

Concatenated name.

Concatenated the groupName and name and then returns the combination.
return groupName+":"+name

os::smart_ptr<os::smart_ptr<**nodeNameReference**> > crypto::nodeGroup::namesByTimestamp (unsigned int & size)

Returns names sorted by timestamp.

Parameters

out	*size*	Size of array to be returned

Returns

crypto::nodeGroup::sortedNames (p. 159)

unsigned int crypto::nodeGroup::numberOfKeys ( ) const [inline]

Returns the number of keys.

Returns

crypto::nodeGroup::keyList.size()

unsigned int crypto::nodeGroup::numberOfNames ( ) const [inline]

Returns the number of names.

Returns

crypto::nodeGroup::nameList.size()

void crypto::nodeGroup::sortKeys ( ) [private]

Sorts keys by timestamp.

void crypto::nodeGroup::sortNames ( ) [private]

Sorts names by timestamp.

6.28.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Only **keyBank** (p. 134) can load a node group.

6.28.5 Member Data Documentation

keyBank* crypto::nodeGroup::_master [private]

Pointer to key bank.

os::asyncAVLTree<**nodeKeyReference**> crypto::nodeGroup::keyList [private]

List of all keys associated with this node.

os::asyncAVLTree<**nodeNameReference**> crypto::nodeGroup::nameList [private]

List of all names associated with this node.

os::smart_ptr<os::smart_ptr<**nodeKeyReference**> > crypto::nodeGroup::sortedKeys [private]

Array of keys sorted by timestamp.

os::smart_ptr<os::smart_ptr<**nodeNameReference**> > crypto::nodeGroup::sortedNames [private]

Array of names sorted by timestamp.

std::mutex crypto::nodeGroup::sortingLock [private]

Lock used for sorting.

6.29 crypto::nodeKeyReference Class Reference

Key storage node.

Public Member Functions

- virtual ~**nodeKeyReference** ()
  Virtual destructor.
- **nodeGroup** * **master** ()
  Returns a pointer to its master.
- os::smart_ptr< **number** > **key** () const
  Returns the key.
- uint16_t **algoID** () const
  Returns the algorithm key.
- uint16_t **keySize** () const
  Returns the key size.
- uint64_t **timestamp** () const
  Returns the timestamp.
- int **compare** (const **nodeKeyReference** &comp) const
  *Compare **crypto::nodeKeyReference*** (p. 159).
- bool **operator==** (const **nodeKeyReference** &comp) const
  Equality operator.
- bool **operator!=** (const **nodeKeyReference** &comp) const
  Not-equals operator.
- bool **operator**> (const **nodeKeyReference** &comp) const
  Greater-than operator.
- bool **operator**>**=** (const **nodeKeyReference** &comp) const
  Greater-than/equals to operator.

- bool **operator**< (const **nodeKeyReference** &comp) const

  Less-than operator.
- bool **operator**<**=** (const **nodeKeyReference** &comp) const

  Less-than/equals to operator.

Private Member Functions

- **nodeKeyReference** (**nodeGroup** ∗**master**, os::smart_ptr< **number** > **key**, uint16_t **algoID**, uint16_t **keySize**, uint64_t **timestamp**=os::getTimestamp())

  Key reference node constructor.
- **nodeKeyReference** (os::smart_ptr< **number** > **key**, uint16_t **algoID**, uint16_t **keySize**)

  Key reference node constructor for searching.

Private Attributes

- **nodeGroup** ∗ **_master**

  Pointer to node group.
- os::smart_ptr< **number** > **_key**

  Shared pointer to public key.
- uint16_t **_algoID**

  ID of public key algorithm.
- uint16_t **_keySize**

  Size of public key.
- uint64_t **_timestamp**

  Timestamp key created.

Friends

- class **nodeGroup**

  *Friendship with **crypto::nodeGroup** (p. 153).*
- class **keyBank**

  *Friendship with **crypto::keyBank** (p. 134).*

6.29.1 Detailed Description

Key storage node.

Allows for storage and sorting of a node group by its key. This node holds a reference to the larger group node.

6.29.2 Constructor & Destructor Documentation

crypto::nodeKeyReference::nodeKeyReference (**nodeGroup** ∗ master, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp = `os::getTimestamp()`) `[private]`

Key reference node constructor.

Parameters

	[in/out]	master Reference to the 'master' group
in	*key*	The public key of a given node
in	*algoID*	The algorithm identifier
in	*keySize*	Size of the key provided
	timestamp	The time this node was created, 'now' by defult

crypto::nodeKeyReference::nodeKeyReference (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [private]

Key reference node constructor for searching.

Parameters

in	*key*	The public key of a given node
in	*algoID*	The algorithm identifier
in	*keySize*	Size of the key provided

virtual crypto::nodeKeyReference::~nodeKeyReference ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.29.3 Member Function Documentation

uint16_t crypto::nodeKeyReference::algoID ( ) const [inline]

Returns the algorithm key.

Returns

crypto::nodeKeyReference::_algoID (p. 164)

int crypto::nodeKeyReference::compare (const **nodeKeyReference** & comp) const

Compare **crypto::nodeKeyReference** (p. 159).

Compares two node key references by their public key, returning the result in the form of a 1,0 or -1.

Parameters

in	*comp*	Key reference to compare against

Returns

1, 0, -1 (Greater than, equal to, less than)

os::smart_ptr<**number**> crypto::nodeKeyReference::key ( ) const [inline]

Returns the key.

Returns

crypto::nodeKeyReference::_key (p. 164)

uint16_t crypto::nodeKeyReference::keySize ( ) const [inline]

Returns the key size.

Returns

crypto::nodeKeyReference::_keySize (p. 164)

nodeGroup* crypto::nodeKeyReference::master ( ) [inline]

Returns a pointer to its master.

Returns

crypto::~nodeKeyReference::_master

bool crypto::nodeKeyReference::operator!= (const **nodeKeyReference** & comp) const [inline]

Not-equals operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if not equal, else, false

bool crypto::nodeKeyReference::operator< (const **nodeKeyReference** & comp) const [inline]

Less-than operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if less than, else, false

bool crypto::nodeKeyReference::operator<= (const **nodeKeyReference** & comp) const `[inline]`

Less-than/equals to operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if less than or equal to, else, false

bool crypto::nodeKeyReference::operator== (const **nodeKeyReference** & comp) const `[inline]`

Equality operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if equal, else, false

bool crypto::nodeKeyReference::operator> (const **nodeKeyReference** & comp) const `[inline]`

Greater-than operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if greater than, else, false

bool crypto::nodeKeyReference::operator>= (const **nodeKeyReference** & comp) const `[inline]`

Greater-than/equals to operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if greater than or equal to, else, false

uint64_t crypto::nodeKeyReference::timestamp ( ) const [inline]

Returns the timestamp.

Returns

crypto::nodeKeyReference::_timestamp (p. 164)

6.29.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Friendship with **crypto::keyBank** (p. 134).
The key bank must be able to create a node key to search by key

friend class **nodeGroup** [friend]

Friendship with **crypto::nodeGroup** (p. 153).
Only node groupings can meaningfully create this class, so the constructor is private and only accessable by **crypto::nodeGroup** (p. 153).

6.29.5 Member Data Documentation

uint16_t crypto::nodeKeyReference::_algoID [private]

ID of public key algorithm.

os::smart_ptr<**number**> crypto::nodeKeyReference::_key [private]

Shared pointer to public key.

uint16_t crypto::nodeKeyReference::_keySize [private]

Size of public key.

nodeGroup* crypto::nodeKeyReference::_master [private]

Pointer to node group.

uint64_t crypto::nodeKeyReference::_timestamp [private]

Timestamp key created.

6.30 crypto::nodeNameReference Class Reference

Name storage node.

Public Member Functions

- virtual ~**nodeNameReference** ()

  Virtual destructor.
- **nodeGroup** * **master** ()

  Returns a pointer to its master.
- std::string **groupName** () const

  Returns the group name.
- std::string **name** () const

  Returns the name.
- uint64_t **timestamp** () const

  Returns the timestamp.
- int **compare** (const **nodeNameReference** &comp) const

  *Compare **crypto::nodeNameReference** (p. 165).*
- bool **operator==** (const **nodeNameReference** &comp) const

  Equality operator.
- bool **operator!=** (const **nodeNameReference** &comp) const

  Not-equals operator.
- bool **operator**> (const **nodeNameReference** &comp) const

  Greater-than operator.
- bool **operator**>**=** (const **nodeNameReference** &comp) const

  Greater-than/equals to operator.
- bool **operator**< (const **nodeNameReference** &comp) const

  Less-than operator.
- bool **operator**<**=** (const **nodeNameReference** &comp) const

  Less-than/equals to operator.

Private Member Functions

- **nodeNameReference** (**nodeGroup** *__master__, std::string **groupName**, std::string **name**, uint64_t **timestamp**=os::getTimestamp())

  Name reference node constructor.
- **nodeNameReference** (std::string **groupName**, std::string **name**)

  Name reference node constructor for searching.

Private Attributes

- **nodeGroup** * **_master**

  Pointer to node group.
- std::string **_groupName**

  Name of the group this name is from.
- std::string **_name**

  Name of the node.
- uint64_t **_timestamp**

  Timestamp key created.

Friends

- class **nodeGroup**

  *Friendship with **crypto::nodeGroup** (p. 153).*
- class **keyBank**

  *Friendship with **crypto::keyBank** (p. 134).*

6.30.1 Detailed Description

Name storage node.

Allows for storage and sorting of a node group by its name. This node holds a reference to the larger group node.

6.30.2 Constructor & Destructor Documentation

crypto::nodeNameReference::nodeNameReference (**nodeGroup** * master, std::string groupName, std::string name, uint64_t timestamp = `os::getTimestamp()`) `[private]`

Name reference node constructor.

Parameters

	[in/out]	master Reference to the 'master' group
`in`	*groupName*	Group name of the node being registered
`in`	*name*	Name of the node being registered
	timestamp	The time this node was created, 'now' by defult

crypto::nodeNameReference::nodeNameReference (std::string groupName, std::string name) `[private]`

Name reference node constructor for searching.

Parameters

`in`	*groupName*	Group name of the node being registered

Parameters

in	*name*	Name of the node being registered

virtual crypto::nodeNameReference::~nodeNameReference ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.30.3 Member Function Documentation

int crypto::nodeNameReference::compare (const **nodeNameReference** & comp) const

Compare **crypto::nodeNameReference** (p. 165).

Compares two node name references by their group and name, returning the result in the form of a 1,0 or -1.

Parameters

in	*comp*	Name reference to compare against

Returns

1, 0, -1 (Greater than, equal to, less than)

std::string crypto::nodeNameReference::groupName ( ) const [inline]

Returns the group name.

Returns

crypto::nodeNameReference::_groupName (p. 170)

nodeGroup* crypto::nodeNameReference::master ( ) [inline]

Returns a pointer to its master.

Returns

crypto::nodeNameReference::_master (p. 170)

std::string crypto::nodeNameReference::name ( ) const [inline]

Returns the name.

Returns

crypto::nodeNameReference::_name (p. 170)

bool crypto::nodeNameReference::operator!= (const **nodeNameReference** & comp) const [inline]

Not-equals operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if not equal, else, false

bool crypto::nodeNameReference::operator< (const **nodeNameReference** & comp) const [inline]

Less-than operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if less than, else, false

bool crypto::nodeNameReference::operator<= (const **nodeNameReference** & comp) const [inline]

Less-than/equals to operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if less than or equal to, else, false

bool crypto::nodeNameReference::operator== (const **nodeNameReference** & comp) const [inline]

Equality operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if equal, else, false

bool crypto::nodeNameReference::operator> (const **nodeNameReference** & comp) const [inline]

Greater-than operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if greater than, else, false

bool crypto::nodeNameReference::operator>= (const **nodeNameReference** & comp) const [inline]

Greater-than/equals to operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if greater than or equal to, else, false

uint64_t crypto::nodeNameReference::timestamp ( ) const [inline]

Returns the timestamp.

Returns

crypto::nodeNameReference::_timestamp (p. 170)

6.30.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Friendship with **crypto::keyBank** (p. 134).

The key bank must be able to create a node name to search by name

friend class **nodeGroup** [friend]

Friendship with **crypto::nodeGroup** (p. 153).

Only node groupings can meaningfully create this class, so the constructor is private and only accessable by **crypto::nodeGroup** (p. 153).

6.30.5 Member Data Documentation

std::string crypto::nodeNameReference::_groupName [private]

Name of the group this name is from.

nodeGroup* crypto::nodeNameReference::_master [private]

Pointer to node group.

std::string crypto::nodeNameReference::_name [private]

Name of the node.

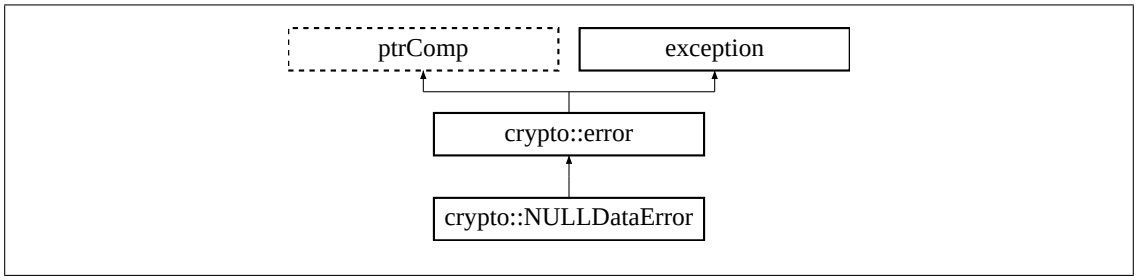
uint64_t crypto::nodeNameReference::_timestamp [private]

Timestamp key created.

6.31 crypto::NULLDataError Class Reference

NULL data error.

Inheritance diagram for crypto::NULLDataError:



Public Member Functions

- virtual ~**NULLDataError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "NULL Data".
- std::string **errorDescription** () const

  Long error descriptor Returns "A function was passed NULL data where this is illegal".

6.31.1 Detailed Description

NULL data error.

Thrown when NULL data is passed to a function or class.

6.31.2 Constructor & Destructor Documentation

virtual crypto::NULLDataError::~NULLDataError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.31.3 Member Function Documentation

std::string crypto::NULLDataError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "A function was passed NULL data where this is illegal".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::NULLDataError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "NULL Data".

Returns

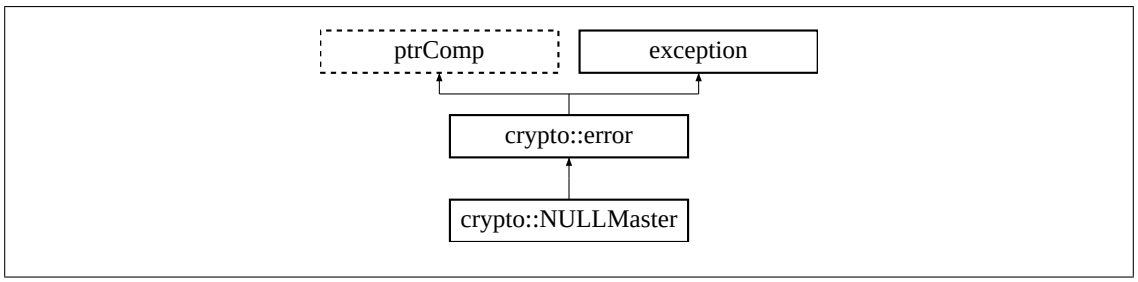
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.32 crypto::NULLMaster Class Reference

NULL master error.

Inheritance diagram for crypto::NULLMaster:



Public Member Functions

- virtual ~**NULLMaster** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "NULL Master pointer".
- std::string **errorDescription** () const

  Long error descriptor Returns "A class received a NULL master pointer, this is illegal".

6.32.1 Detailed Description

NULL master error.

Thrown when a class is passed a NULL master where such a class must have a defined master.

6.32.2 Constructor & Destructor Documentation

virtual crypto::NULLMaster::~NULLMaster ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.32.3 Member Function Documentation

std::string crypto::NULLMaster::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "A class received a NULL master pointer, this is illegal".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::NULLMaster::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "NULL Master pointer".

Returns

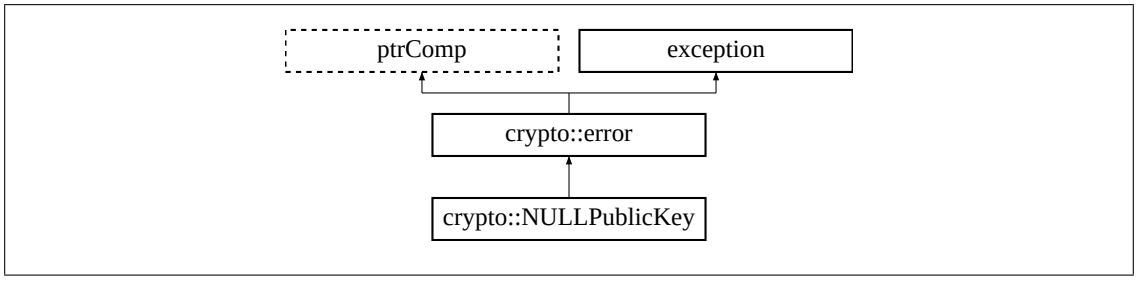
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.33 crypto::NULLPublicKey Class Reference

NULL public-key error.

Inheritance diagram for crypto::NULLPublicKey:



Public Member Functions

- virtual ~**NULLPublicKey** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Public Key NULL".
- std::string **errorDescription** () const

  Long error descriptor Returns "Attempted to bind a public key of illegal type NULL".

6.33.1 Detailed Description

NULL public-key error.

Thrown when a NULL public-key or public-key of undefined type is used.

6.33.2 Constructor & Destructor Documentation

virtual crypto::NULLPublicKey::~NULLPublicKey ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.33.3 Member Function Documentation

std::string crypto::NULLPublicKey::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Attempted to bind a public key of illegal type NULL".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::NULLPublicKey::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Public Key NULL".

Returns

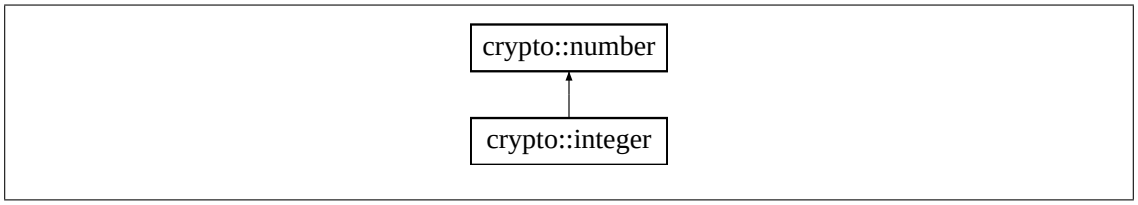
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.34 crypto::number Class Reference

Basic number definition.

Inheritance diagram for crypto::number:



Public Member Functions

- **number** (struct **numberType** *numDef=**buildNullNumberType**())

  Construct with number definition.
- **number** (uint16_t **size**, struct **numberType** *numDef=**buildNullNumberType**())

  Construct with size.
- **number** (const uint32_t *d, uint16_t **size**, struct **numberType** *numDef=**buildNullNumber↩ Type**())

  Construct with data array.
- **number** (const **number** &num)

  Copy constructor.
- **number** & **operator=** (const **number** &num)

  Equality constructor.
- virtual ~**number** ()

  Virtual destructor.
- void **reduce** ()

  Eliminate high-order zeros.
- void **expand** (uint16_t **size**)

  Expand number size.
- os::smart_ptr< unsigned char > **getCharData** (unsigned int &arr_len) const

  Build byte array.
- os::smart_ptr< unsigned char > **getCompCharData** (unsigned int &arr_len) const

  Build compatibility byte array.
- std::string **toString** () const

  Build hex string from number.
- void **fromString** (const std::string &str)

  Re-builds number from provided string.
- uint32_t **operator[ ]** (uint16_t pos) const

  Read-only data access.
- uint32_t & **operator[ ]** (uint16_t pos)

  Read/write data access.
- const bool **operator==** (const **number** &comp) const

  '==' comparison operator
- const bool **operator!=** (const **number** &comp) const

  '!=' comparison operator
- const bool **operator**<**=** (const **number** &comp) const

'<=' comparison operator

- const bool **operator>=** (const **number** &comp) const

  '>=' comparison operator
- const bool **operator<** (const **number** &comp) const

  '<' comparison operator
- const bool **operator>** (const **number** &comp) const

  '>' comparison operator
- int **compare** (const **number** *n2) const

  Compares two numbers.
- void **addition** (const **number** *n2, **number** *result) const

  Addition function.
- void **subtraction** (const **number** *n2, **number** *result) const

  Subtraction function.
- void **rightShift** (uint16_t n2, **number** *result) const

  Right shift function.
- void **leftShift** (uint16_t n2, **number** *result) const

  Left shift function.
- void **multiplication** (const **number** *n2, **number** *result) const

  Multiplication function.
- void **division** (const **number** *n2, **number** *result) const

  Division function.
- void **modulo** (const **number** *n2, **number** *result) const

  Modulo function.
- void **exponentiation** (const **number** *n2, **number** *result) const

  Exponentiation function.
- void **moduloExponentiation** (const **number** *n2, const **number** *n3, **number** *result) const

  Modular exponentiation.
- void **gcd** (const **number** *n2, **number** *result) const

  Greatest-common-denominator function.
- void **modInverse** (const **number** *n2, **number** *result) const

  Modular-inverse function.
- **number operator|** (const **number** &op) const

  Or operator.
- **number** & **operator|=** (const **number** &op)

  Or-equals operator.
- **number operator&** (const **number** &op) const

  And operator.
- **number** & **operator&=** (const **number** &op)

  And-equals operator.
- **number operator^** (const **number** &op) const

  X-Or operator.
- **number** & **operator^=** (const **number** &op)

  X-Or-equals operator.

- **number operator**~ () const

  Negate operator.
- virtual bool **checkType** () const

  Check if the number is valid.
- bool **hasCompare** () const

  Check for the 'compare' function.
- bool **hasAddition** () const

  Check for the 'addition' function.
- bool **hasSubtraction** () const

  Check for the 'subtraction' function.
- bool **hasRightShift** () const

  Check for the 'rightShift' function.
- bool **hasLeftShift** () const

  Check for the 'leftShift' function.
- bool **hasMultiplication** () const

  Check for the 'multiplication' function.
- bool **hasDivision** () const

  Check for the 'division' function.
- bool **hasModulo** () const

  Check for the 'modulo' function.
- bool **hasExponentiation** () const

  Check for the 'exponentiation' function.
- bool **hasModuloExponentiation** () const

  Check for the 'moduloExponentiation' function.
- bool **hasGCD** () const

  Check for the 'gcd' function.
- bool **hasModInverse** () const

  Check for the 'modInverse' function.
- uint16_t **size** () const

  Access data size.
- uint32_t ∗ **data** ()

  Data access.
- const uint32_t ∗ **data** () const

  Constant data access.
- const struct **numberType** ∗ **numberDefinition** () const

  Access number definition.
- int **typeID** () const

  Access number ID.
- std::string **name** () const

  Access number name.

Protected Member Functions

- int **_compare** (const **number** &n2) const

  Compares two numbers.

Protected Attributes

- struct **numberType** * **_numDef**

  Definition of number algorithms.
- uint16_t **_size**

  Size of the data array.
- uint32_t * **_data**

  Data array.

6.34.1 Detailed Description

Basic number definition.

This class defines the basics of all large number classes. Operators are specifically defined in each class which inherits from number.

6.34.2 Constructor & Destructor Documentation

crypto::number::number (struct **numberType** * numDef = **buildNullNumberType**())

Construct with number definition.

Parameters

in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (uint16_t size, struct **numberType** * numDef = **buildNullNumberType**())

Construct with size.

Parameters

in	*size*	Size of the number to be constructed
in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (const uint32_t * d, uint16_t size, struct **numberType** * numDef = **buildNullNumberType**())

Construct with data array.

Parameters

in	*d*	Data array to bind to this number
in	*size*	Size of the number to be constructed
in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (const **number** & num)

Copy constructor.

Parameters

in	*num*	Number used to construct this

virtual crypto::number::~number ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.34.3 Member Function Documentation

int crypto::number::_compare (const **number** & n2) const [protected]

Compares two numbers.

Parameters

in	*n2*	Number to be compared against

Returns

0 if equal, 1 if greater than, -1 if less than

void crypto::number::addition (const **number** * n2, **number** * result) const

Addition function.

Preforms this+n2=result. Note that this function will only preform the addition if the number definition defines an addition function.

Parameters

in	*n2*	Number to be added
out	*result*	Result of addition

Returns

void

virtual bool crypto::number::checkType ( ) const [inline], [virtual]

Check if the number is valid.

By default, this function returns false. Numbers which inherit this class are expected to use this function to check if the number definition matches the class definition.

Returns

true if valid type, else, false

Reimplemented in **crypto::integer** (p. 128).

int crypto::number::compare (const **number** * n2) const

Compares two numbers.

Parameters

in	*n2*	Number to be compared against

Returns

0 if equal, 1 if greater than, -1 if less than

uint32_t* crypto::number::data ( ) [inline]

Data access.

Returns

crypto::number::_data (p. 190)

const uint32_t* crypto::number::data ( ) const [inline]

Constant data access.

Returns

crypto::number::_data (p. 190)

void crypto::number::division (const **number** * n2, **number** * result) const

Division function.

Preforms this/n2=result. Note that this function will only preform the division if the number definition defines an division function.

Parameters

in	*n2*	Number to be divided by
out	*result*	Result of division

Returns

void

void crypto::number::expand (uint16_t size)

Expand number size.

Parameters

in	*size*	Size of the number to be constructed

Returns

void

void crypto::number::exponentiation (const **number** * n2, **number** * result) const

Exponentiation function.

Preforms this^n2=result. Note that this function will only preform the exponentiation if the number definition defines an exponentiation function.

Parameters

in	*n2*	Number to be raised to
out	*result*	Result of exponentiation

Returns

void

void crypto::number::fromString (const std::string & str)

Re-builds number from provided string.

Parameters

in	*str*	Hex string representing number

Returns

void

void crypto::number::gcd (const **number** * n2, **number** * result) const

Greatest-common-denominator function.

Preforms GCD of this and n2=result. Note that this function will only preform the greatest-common-denominator if the number definition defines an greatest-common-denominator function.

Parameters

in	*n2*	GCD target
out	*result*	Result of greatest-common-denominator

Returns

void

os::smart_ptr<unsigned char> crypto::number::getCharData (unsigned int & arr_len) const

Build byte array.

Constructs a byte array based on the data array of this number. Useful for binary saving and packet-izing.

Parameters

out	*arr_len*	return Byte array

os::smart_ptr<unsigned char> crypto::number::getCompCharData (unsigned int & arr_len) const

Build compatibility byte array.

Constructs a byte array based on the data array of this number. First eliminates endian differences of operating systems.

Parameters

out	*arr_len*	return Byte array

bool crypto::number::hasAddition ( ) const [inline]

Check for the 'addition' function.

Returns

crypto::number::_numDef (p. 190)->addition

bool crypto::number::hasCompare ( ) const [inline]

Check for the 'compare' function.

Returns

crypto::number::_numDef (p. 190)->compare

bool crypto::number::hasDivision ( ) const [inline]

Check for the 'division' function.

Returns

crypto::number::_numDef (p. 190)->division

bool crypto::number::hasExponentiation ( ) const [inline]

Check for the 'exponentiation' function.

Returns

crypto::number::_numDef (p. 190)->exponentiation

bool crypto::number::hasGCD ( ) const [inline]

Check for the 'gcd' function.

Returns

crypto::number::_numDef (p. 190)->gcd

bool crypto::number::hasLeftShift ( ) const [inline]

Check for the 'leftShift' function.

Returns

crypto::number::_numDef (p. 190)->leftShift

bool crypto::number::hasModInverse ( ) const [inline]

Check for the 'modInverse' function.

Returns

crypto::number::_numDef (p. 190)->modInverse

bool crypto::number::hasModulo ( ) const [inline]

Check for the 'modulo' function.

Returns

crypto::number::_numDef (p. 190)->modulo

bool crypto::number::hasModuloExponentiation ( ) const [inline]

Check for the 'moduloExponentiation' function.

Returns

crypto::number::_numDef (p. 190)->moduloExponentiation

bool crypto::number::hasMultiplication ( ) const [inline]

Check for the 'multiplication' function.

Returns

crypto::number::_numDef (p. 190)->multiplication

bool crypto::number::hasRightShift ( ) const [inline]

Check for the 'rightShift' function.

Returns

crypto::number::_numDef (p. 190)->rightShift

bool crypto::number::hasSubtraction ( ) const [inline]

Check for the 'subtraction' function.

Returns

crypto::number::_numDef (p. 190)->subtraction

void crypto::number::leftShift (uint16_t n2, **number** * result) const

Left shift function.

Preforms this<<n2=result. Note that this function will only preform the shift if the number definition defines an leftShift function.

Parameters

in	*n2*	Bits to be shifted by
out	*result*	Result of shift

Returns

void

void crypto::number::modInverse (const **number** * n2, **number** * result) const

Modular-inverse function.

Preforms (this^-1)n2=result. Note that this function will only preform the modular-inverse if the number definition defines an modular-inverse function.

Parameters

in	*n2*	Number which defines the modulo space
out	*result*	Result of modular-inverse

Returns

void

void crypto::number::modulo (const **number** * n2, **number** * result) const

Modulo function.

Preforms thisn2=result. Note that this function will only preform the modulo if the number definition defines an modulo function.

Parameters

in	*n2*	Number to be moded by
out	*result*	Result of modulo

Returns

void

void crypto::number::moduloExponentiation (const **number** * n2, const **number** * n3, **number** * result) const

Modular exponentiation.

Preforms this^n2 n3=result. Note that this function will only preform the modular exponentiation if the number definition defines an modular exponentiation function.

Parameters

in	*n2*	Number to be raised to
in	*n3*	Number defines modulo space
out	*result*	Result of exponentiation

Returns

void

void crypto::number::multiplication (const **number** * n2, **number** * result) const

Multiplication function.

Preforms this*n2=result. Note that this function will only preform the multiplication if the number definition defines an multiplication function.

Parameters

in	*n2*	Number to be multiplied
out	*result*	Result of multiplication

Returns

void

std::string crypto::number::name ( ) const [inline]

Access number name.

Returns

crypto::number::_numDef (p. 190)->name

const struct **numberType*** crypto::number::numberDefinition ( ) const [inline]

Access number definition.

Returns

crypto::number::_numDef (p. 190)

const bool crypto::number::operator!= (const **number** & comp) const

'!=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this != comp

number crypto::number::operator& (const **number** & op) const

And operator.

Preforms bitwise and on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this & op

number& crypto::number::operator&= (const **number** & op)

And-equals operator.

Preforms bitwise and-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this = this & op

const bool crypto::number::operator< (const **number** & comp) const

'<' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this < comp

const bool crypto::number::operator<= (const **number** & comp) const

'<=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this <= comp

number& crypto::number::operator= (const **number** & num)

Equality constructor.

Parameters

in	*num*	Number used to re-build this

Returns

Reference to this

const bool crypto::number::operator== (const **number** & comp) const

'==' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this == comp

const bool crypto::number::operator> (const **number** & comp) const

'>' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this > comp

const bool crypto::number::operator>= (const **number** & comp) const

'>=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this >= comp

uint32_t crypto::number::operator[ ] (uint16_t pos) const

Read-only data access.

Parameters

in	*pos*	Index to access

Returns

crypto::number::_data (p. 190)[pos]

uint32_t& crypto::number::operator[ ] (uint16_t pos)

Read/write data access.

Parameters

in	*pos*	Index to access

Returns

crypto::number::_data (p. 190)[pos]

number crypto::number::operator^ (const **number** & op) const

X-Or operator.

Preforms bitwise exclusive-or on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this ^ op

number& crypto::number::operator^= (const **number** & op)

X-Or-equals operator.

Preforms bitwise exclusive-or-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this=this ^ op

number crypto::number::operator| (const **number** & op) const

Or operator.

Preforms bitwise or on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this | op

number& crypto::number::operator|= (const **number** & op)

Or-equals operator.

Preforms bitwise or-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this = this | op

number crypto::number::operator~ ( ) const

Negate operator.

Flips all bits in the number, returning a new number.

Returns

~this

void crypto::number::reduce ( )

Eliminate high-order zeros.

Returns

void

void crypto::number::rightShift (uint16_t n2, **number** * result) const

Right shift function.

Preforms this>>n2=result. Note that this function will only preform the shift if the number definition defines an rightShift function.

Parameters

in	*n2*	Bits to be shifted by
out	*result*	Result of shift

Returns

void

uint16_t crypto::number::size ( ) const [inline]

Access data size.

Returns

crypto::number::_size (p. 190)

void crypto::number::subtraction (const **number** * n2, **number** * result) const

Subtraction function.
Preforms this-n2=result. Note that this function will only preform the subtraction if the number definition defines an subtraction function.

Parameters

in	*n2*	Number to be subtracted
out	*result*	Result of subtraction

Returns

void

std::string crypto::number::toString ( ) const

Build hex string from number.

Returns

Hex string

int crypto::number::typeID ( ) const [inline]

Access number ID.

Returns

crypto::number::_numDef (p. 190)->typeID

6.34.4 Member Data Documentation

uint32_t* crypto::number::_data [protected]

Data array.

struct **numberType*** crypto::number::_numDef [protected]

Definition of number algorithms.

uint16_t crypto::number::_size [protected]

Size of the data array.

6.35 numberType Struct Reference

Number type function structure.

Public Attributes

- int **typeID**

  ID integer of the number type.
- const char * **name**

  Name of the number type.
- **compareFunction compare**

  Pointer to comparison function.
- **operatorFunction addition**

  Pointer to addition function.
- **operatorFunction subtraction**

  Pointer to subtraction function.
- **shiftFunction rightShift**

  Pointer to right-shift function.
- **shiftFunction leftShift**

  Pointer to left-shift function.
- **operatorFunction multiplication**

  Pointer to multiplication function.
- **operatorFunction division**

  Pointer to division function.
- **operatorFunction modulo**

  Pointer to modulo function.
- **operatorFunction exponentiation**

  Pointer to exponentiation function.
- **tripleCalculation moduloExponentiation**

  Pointer to modulo exponentiation function.
- **operatorFunction gcd**

  Pointer to greatest common denominator function.
- **operatorFunction modInverse**

  Pointer to modulo inverse function.

6.35.1 Detailed Description

Number type function structure.

This structure contains a series of meaningful function pointers which define functions required to meaningfully define a numerical system.

6.35.2 Member Data Documentation

operatorFunction numberType::addition

Pointer to addition function.

compareFunction numberType::compare

Pointer to comparison function.

operatorFunction numberType::division

Pointer to division function.

operatorFunction numberType::exponentiation

Pointer to exponentiation function.

operatorFunction numberType::gcd

Pointer to greatest common denominator function.

shiftFunction numberType::leftShift

Pointer to left-shift function.

operatorFunction numberType::modInverse

Pointer to modulo inverse function.

operatorFunction numberType::modulo

Pointer to modulo function.

tripleCalculation numberType::moduloExponentiation

Pointer to modulo exponentiation function.

operatorFunction numberType::multiplication

Pointer to multiplication function.

const char* numberType::name

Name of the number type.

shiftFunction numberType::rightShift

Pointer to right-shift function.

operatorFunction numberType::subtraction

Pointer to subtraction function.

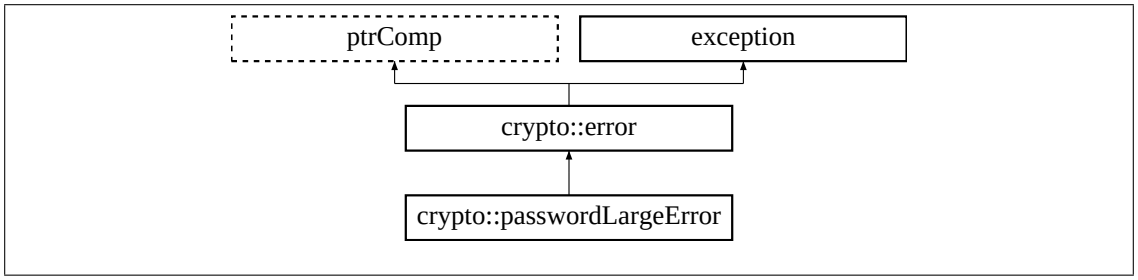
int numberType::typeID

ID integer of the number type.

6.36 crypto::passwordLargeError Class Reference

Symmetric key too big.

Inheritance diagram for crypto::passwordLargeError:



Public Member Functions

- virtual ~**passwordLargeError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Password Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Password too large".

6.36.1 Detailed Description

Symmetric key too big.

Thrown when a symmetric key is provided which is bigger than the maximum for the specific algorithm.

6.36.2 Constructor & Destructor Documentation

virtual crypto::passwordLargeError::~passwordLargeError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.36.3 Member Function Documentation

std::string crypto::passwordLargeError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Password too large".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::passwordLargeError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Password Size Error".

Returns

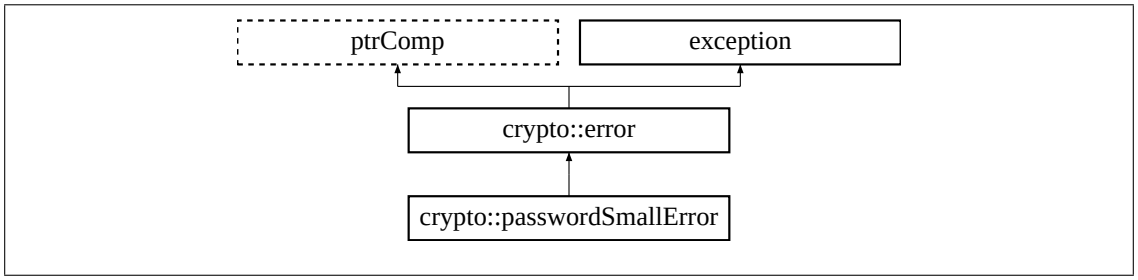
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.37 crypto::passwordSmallError Class Reference

Symmetric key too small.

Inheritance diagram for crypto::passwordSmallError:



Public Member Functions

- virtual ~**passwordSmallError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Password Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Password too small".

6.37.1 Detailed Description

Symmetric key too small.

Thrown when a symmetric key is provided which is smaller than the minimum for the specific algorithm.

6.37.2 Constructor & Destructor Documentation

virtual crypto::passwordSmallError::~passwordSmallError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.37.3 Member Function Documentation

std::string crypto::passwordSmallError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Password too small".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::passwordSmallError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Password Size Error".

Returns

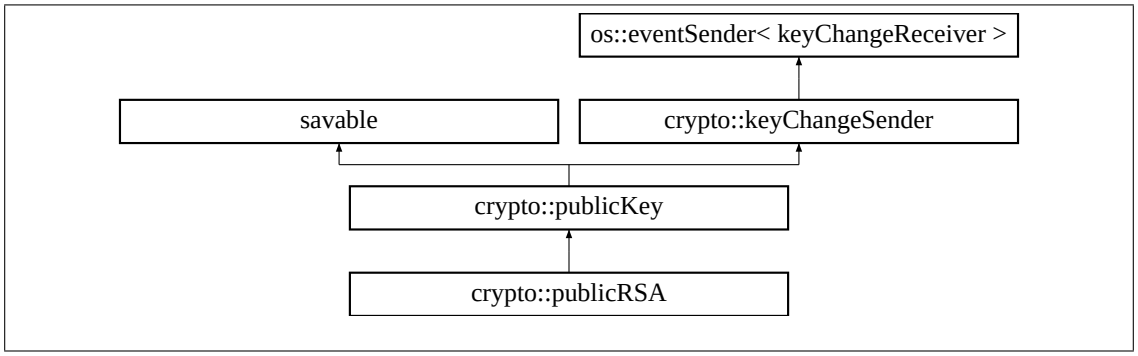
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.38 crypto::publicKey Class Reference

Base public-key class.

Inheritance diagram for crypto::publicKey:



Public Member Functions

- virtual ~**publicKey** ()

  Virtual destructor.
- bool **searchKey** (**hash** hsh, unsigned int &hist, bool &type)

  Searches for key by hash.
- bool **searchKey** (os::smart_ptr< **number** > key, unsigned int &hist, bool &type)

  Searches for key.
- virtual os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num) const

  Converts number to correct type.
- virtual os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len) const

  Converts array to correct number type.
- virtual os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len) const

  Converts byte array to correct number type.
- os::smart_ptr< **number** > **getN** () const

  Public key access.
- os::smart_ptr< **number** > **getD** () const

  Private key access.
- uint64_t **timestamp** () const

  Time-stamp access.
- os::smart_ptr< **number** > **getOldN** (unsigned int **history**=0)

  Access old public keys.
- os::smart_ptr< **number** > **getOldD** (unsigned int **history**=0)

  Access old private keys.
- uint64_t **getOldTimestamp** (unsigned int **history**=0)

  Access old time-stamps.
- virtual void **generateNewKeys** ()

  Key generation function.
- virtual bool **generating** ()

  Tests if the keys are in the process of generating.
- uint16_t **algorithm** () const

  Access algorithm ID.
- virtual std::string **algorithmName** () const

  Access algorithm name.
- uint16_t **size** () const

  Access key size.
- void **readLock** ()

  Increment read lock.
- void **readUnlock** ()

  Decrement read lock.
- void **setHistory** (uint16_t hist)

  Sets history size.

- uint16_t **history** () const
- void **save** ()

  Re-save the entire structure.
- void **loadFile** ()

  Loads the structure from a file.
- void **setFileName** (std::string **fileName**)

  Set the save file name.
- void **setPassword** (unsigned char *key, unsigned int keyLen)

  Binds a new symmetric key.
- void **setPassword** (std::string password)
- void **setEncryptionAlgorithm** (os::smart_ptr< **streamPackageFrame** > stream_algo)

  Sets the symmetric encryption algorithm.
- const std::string & **fileName** () const

  Return the save file path.
- void **addKeyPair** (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint64_t tms=os↩ ::getTimestamp())

  Add key pair.
- virtual os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN=NULL) const

  Number encode.
- virtual void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN=NULL) const

  Data encode against number.
- virtual void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const

  Data encode.
- virtual os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code) const

  Number decode.
- virtual os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code, unsigned int hist)

  Number decode, old key.
- void **decode** (unsigned char *code, unsigned int codeLength) const

  Data decode.
- void **decode** (unsigned char *code, unsigned int codeLength, unsigned int hist)

  Data decode, old key.
- bool **operator==** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator!=** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**< (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**> (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**<**=** (const **publicKey** &cmp) const

Compares equality by size and algorithm.

- bool **operator>=** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.

Static Public Member Functions

- static os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num, uint16_t **size**)

  Converts number to correct type, statically.
- static os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len, uint16_t **size**)

  Converts array to correct number type, statically.
- static os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len, uint16_t **size**)

  Converts byte array to correct number type, statically.
- static uint16_t **staticAlgorithm** ()

  Access algorithm ID.
- static std::string **staticAlgorithmName** ()

  Access algorithm name.
- static os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Static number encode.
- static void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Hybrid data encode against number.
- static void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength, uint16_t **size**)

  Static data encode.

Static Public Attributes

- static const unsigned int **CURRENT_INDEX** = ~0

  Current key index Allows the current key to be accessed as historical index '-1'.
- static const bool **PUBLIC** =true

  Public boolean marker.
- static const bool **PRIVATE** =false

  Private boolean marker.
- static const bool **N_MARKER** =true

  N (public) boolean marker.
- static const bool **D_MARKER** =false

  D (private) boolean marker.

Protected Member Functions

- **publicKey** (uint16_t algo, uint16_t sz=size::public512)

  No key constructor.
- **publicKey** (const **publicKey** &ky)

  Copy constructor.
- **publicKey** (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint16_t algo, uint16↩ _t sz=size::public512, uint64_t tms=os::getTimestamp())

  Construct with keys.
- **publicKey** (uint16_t algo, std::string **fileName**, std::string password="", os::smart_ptr< **stream**↩ **PackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- **publicKey** (uint16_t algo, std::string **fileName**, unsigned char ∗key, unsigned int keyLen, os↩ ::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- void **writeLock** ()

  Locks the write lock.
- void **writeUnlock** ()

  Unlocks the write lock.
- int **compare** (const **publicKey** &cmp) const

  Compare this with another public key.
- void **pushOldKeys** (os::smart_ptr< **number** > **n**, os::smart_ptr< **number** > **d**, uint64_t ts)

  Bind old keys to history.

Protected Attributes

- os::smart_ptr< **number** > **n**
- os::smart_ptr< **number** > **d**
- uint64_t **_timestamp**
- os::unsortedList< **number** > **oldN**
- os::unsortedList< **number** > **oldD**
- os::unsortedList< uint64_t > **_timestamps**

Private Attributes

- uint16_t **_size**
- uint16_t **_algorithm**
- uint16_t **_history**
- unsigned char ∗ **_key**

  Symmetric key for encryption.
- unsigned int **_keyLen**

  Length of symmetric key.
- os::smart_ptr< **streamPackageFrame** > **fePackage**
- std::string **_fileName**
- os::multiLock **keyLock**

6.38.1 Detailed Description

Base public-key class.

Class which defines the general structure of a public-private key pair. The class does not define the specifics of the algorithm.

6.38.2 Constructor & Destructor Documentation

crypto::publicKey::publicKey (uint16_t algo, uint16_t sz = `size::public512`) `[protected]`

No key constructor.

Parameters

algo	Algorithm ID
sz	Size of key, size::public512 by default

crypto::publicKey::publicKey (const **publicKey** & ky) `[protected]`

Copy constructor.

Parameters

ky	Public key to be copied

crypto::publicKey::publicKey (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint16_t algo, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`) `[protected]`

Construct with keys.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
algo	Algorithm ID
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicKey::publicKey (uint16_t algo, std::string fileName, std::string password = "", os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`) `[protected]`

Construct with path to file and password.

Parameters

algo	Algorithm ID
fileName	Name of file to find keys
password	String representing symmetric key, "" by default
stream_algo	Symmetric key encryption algorithm, NULL by default

crypto::publicKey::publicKey (uint16_t algo, std::string fileName, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`) `[protected]`

Construct with path to file and password.

Parameters

algo	Algorithm ID
fileName	Name of file to find keys
key	Symmetric key
keyLen	Length of symmetric key
stream_algo	Symmetric key encryption algorithm, NULL by default

virtual crypto::publicKey::~publicKey ( ) `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.38.3 Member Function Documentation

void crypto::publicKey::addKeyPair (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint64_t tms = `os::getTimestamp()`)

Add key pair.

Adds a key-pair and binds the current keys to the history;.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
tms	Time-stamp of the current keys, now by default

Returns

void

uint16_t crypto::publicKey::algorithm ( ) const [inline]

Access algorithm ID.

Returns

crypto::publicKey::_algorithm (p. 214)

virtual std::string crypto::publicKey::algorithmName ( ) const [inline], [virtual]

Access algorithm name.

Returns

crypto::publicKey::staticAlgorithmName() (p. 214)

Reimplemented in **crypto::publicRSA** (p. 228).

int crypto::publicKey::compare (const **publicKey** & cmp) const [protected]

Compare this with another public key.

Compares based on the algorithm ID and size of the key. Note that this will return 0 if two public keys have the same algorithm ID and size even if they have different keys.

Parameters

in	*cmp*	Public key to compare against

Returns

0 if equal, 1 if greater than, -1 if less than

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const os::smart_ptr< **number** > num) const [virtual]

Converts number to correct type.

Parameters

in	*num*	Number to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 228).

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const uint32_t * arr, uint16_t len) const [virtual]

Converts array to correct number type.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 229).

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const unsigned char * arr, unsigned int len) const [virtual]

Converts byte array to correct number type.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 229).

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const os::smart_ptr< **number** > num, uint16_t size) [static]

Converts number to correct type, statically.

Parameters

in	*num*	Number to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const uint32_t * arr, uint16_t len, uint16_t size) [static]

Converts array to correct number type, statically.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const unsigned char * arr, unsigned int len, uint16_t size) [static]

Converts byte array to correct number type, statically.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

virtual os::smart_ptr<**number**> crypto::publicKey::decode (os::smart_ptr< **number** > code) const [virtual]

Number decode.

Uses the private key to decode a set of data. Re-implemented by algorithm definitions which inherit from this class.

Parameters

in	*code*	Data to be decoded

Returns

Decoded number

Reimplemented in **crypto::publicRSA** (p. 230).

virtual os::smart_ptr<**number**> crypto::publicKey::decode (os::smart_ptr< **number** > code, unsigned int hist) [virtual]

Number decode, old key.

Uses the private key to decode a set of data. Re-implemented by algorithm definitions which inherit from this class.

Parameters

in	*code*	Data to be decoded
in	*hist*	Index of historical key

Returns

Decoded number

Reimplemented in **crypto::publicRSA** (p. 231).

void crypto::publicKey::decode (unsigned char * code, unsigned int codeLength) const

Data decode.

Uses the private key to decode a set of data.

Parameters

	[in/out]	code Data to be decoded
in	*codeLength*	Length of code to be decoded

Returns

void

void crypto::publicKey::decode (unsigned char * code, unsigned int codeLength, unsigned int hist)

Data decode, old key.

Uses the private key to decode a set of data.

Parameters

	[in/out]	code Data to be decoded
in	*codeLength*	Length of code to be decoded
in	*hist*	Index of historical key

Returns

void

static os::smart_ptr<**number**> crypto::publicKey::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static number encode.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

Encoded number

static void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Hybrid data encode against number.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against, NULL by default

Returns

void

static void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength, uint16_t size) [static]

Static data encode.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*nLength*	Length of key array
in	*size*	Size of key used

Returns

void

virtual os::smart_ptr<**number**> crypto::publicKey::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN = `NULL`) const `[virtual]`

Number encode.

Parameters

`in`	*code*	Data to be encoded
`in`	*publicN*	Public key to be encoded against, NULL by default

Returns

Encoded number

Reimplemented in **crypto::publicRSA** (p. 232).

virtual void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN = `NULL`) const `[virtual]`

Data encode against number.

Parameters

	[in/out]	code Data to be encoded
`in`	*codeLength*	Length of code array
`in`	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented in **crypto::publicRSA** (p. 232).

virtual void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const `[virtual]`

Data encode.

Parameters

	[in/out]	code Data to be encoded
`in`	*codeLength*	Length of code array
`in`	*publicN*	Public key to be encoded against
`in`	*nLength*	Length of key array

Returns

void

Reimplemented in **crypto::publicRSA** (p. 233).

const std::string& crypto::publicKey::fileName ( ) const [inline]

Return the save file path.

Returns

crypto::publicKey::_fileName (p. 214)

virtual void crypto::publicKey::generateNewKeys ( ) [virtual]

Key generation function.

Generates new keys for the specific algorithm. This is re-implemented by every algorithm.

Returns

void

Reimplemented in **crypto::publicRSA** (p. 233).

virtual bool crypto::publicKey::generating ( ) [inline], [virtual]

Tests if the keys are in the process of generating.

Returns

True if generating new keys

Reimplemented in **crypto::publicRSA** (p. 233).

os::smart_ptr<**number**> crypto::publicKey::getD ( ) const

Private key access.

Returns

crypto::publicKey::d (p. 215)

os::smart_ptr<**number**> crypto::publicKey::getN ( ) const

Public key access.

Returns

crypto::publicKey::n (p. 215)

os::smart_ptr<**number**> crypto::publicKey::getOldD (unsigned int history = 0)

Access old private keys.

Parameters

history	Historical index, 0 by default

Returns

Private key at given index

os::smart_ptr<**number**> crypto::publicKey::getOldN (unsigned int history = `0`)

Access old public keys.

Parameters

history	Historical index, 0 by default

Returns

Public key at given index

uint64_t crypto::publicKey::getOldTimestamp (unsigned int history = `0`)

Access old time-stamps.

Parameters

history	Historical index, 0 by default

Returns

Time-stamp at given index

uint16_t crypto::publicKey::history ( ) const `[inline]`

Access history size

Returns

crypto::publicKey::_history (p. 214)

void crypto::publicKey::loadFile ( )

Loads the structure from a file.

Returns

void

bool crypto::publicKey::operator!= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '!='

bool crypto::publicKey::operator< (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '<'

bool crypto::publicKey::operator<= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '<='

bool crypto::publicKey::operator== (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '=='

bool crypto::publicKey::operator> (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '>'

bool crypto::publicKey::operator>= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '>='

void crypto::publicKey::pushOldKeys (os::smart_ptr< **number** > n, os::smart_ptr< **number** > d, uint64_t ts) [protected]

Bind old keys to history.

Parameters

in	*n*	Old public key
in	*d*	Old private key
in	*ts*	Old time-stamp

Returns

void

void crypto::publicKey::readLock ( ) [inline]

Increment read lock.

Returns

void

void crypto::publicKey::readUnlock ( ) [inline]

Decrement read lock.

Returns

void

void crypto::publicKey::save ( )

Re-save the entire structure.

Returns

void

bool crypto::publicKey::searchKey (**hash** hsh, unsigned int & hist, bool & type)

Searches for key by hash.

Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*hsh*	Hash of the key to be searched for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

True if the key was found, else, false

bool crypto::publicKey::searchKey (os::smart_ptr< **number** > key, unsigned int & hist, bool & type)

Searches for key.

Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*num*	Key to search for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

True if the key was found, else, false

void crypto::publicKey::setEncryptionAlgorithm (os::smart_ptr< **streamPackageFrame** > stream_algo)

Sets the symmetric encryption algorithm.

Parameters

in	*stream_algo*	Symmetric key algorithm

Returns

void

void crypto::publicKey::setFileName (std::string fileName)

Set the save file name.

Parameters

in	*fileName*	Path of save file

Returns

void

void crypto::publicKey::setHistory (uint16_t hist)

Sets history size.

Determines the number of historical keys to keep recorded. Note that keys are sorted by the order they were received into this structure, not their time-stamp.

Parameters

in	*hist*	History size to be bound

Returns

void

void crypto::publicKey::setPassword (unsigned char * key, unsigned int keyLen)

Binds a new symmetric key.

Re-binding of the symmetric key will result in a re-save event through the savable class.

Parameters

in	*key*	Symmetric key
in	*keyLen*	Length of symmetric key

Returns

void

void crypto::publicKey::setPassword (std::string password)

Binds a new symmetric key

Parameters

in	*password*	String representing the symmetric key

Returns

void

uint16_t crypto::publicKey::size ( ) const [inline]

Access key size.

Returns

crypto::publicKey::_size (p. 215)

static uint16_t crypto::publicKey::staticAlgorithm ( ) [inline], [static]

Access algorithm ID.

Returns

crypto::algo::publicNULL

static std::string crypto::publicKey::staticAlgorithmName ( ) [inline], [static]

Access algorithm name.

Returns

"NULL Public Key"

uint64_t crypto::publicKey::timestamp ( ) const [inline]

Time-stamp access.

Returns

crypto::publicKey::_timestamp (p. 215)

void crypto::publicKey::writeLock ( ) [inline], [protected]

Locks the write lock.

Returns

void

void crypto::publicKey::writeUnlock ( ) [inline], [protected]

Unlocks the write lock.

Returns

void

6.38.4 Member Data Documentation

uint16_t crypto::publicKey::_algorithm [private]

@ brief ID of algorithm used

std::string crypto::publicKey::_fileName [private]

@ brief Name of file this key is saved to

uint16_t crypto::publicKey::_history [private]

@ brief Number of historical keys to keep

unsigned char* crypto::publicKey::_key [private]

Symmetric key for encryption.

unsigned int crypto::publicKey::_keyLen [private]

Length of symmetric key.

uint16_t crypto::publicKey::_size [private]

@ brief Size of the keys used

uint64_t crypto::publicKey::_timestamp [protected]

@ brief Date/time keys created

os::unsortedList<uint64_t> crypto::publicKey::_timestamps [protected]

@ brief List of time-stamps for old pairs

const unsigned int crypto::publicKey::CURRENT_INDEX = ~0 [static]

Current key index Allows the current key to be accessed as historical index '-1'.

os::smart_ptr<**number**> crypto::publicKey::d [protected]

@ brief Private key

const bool crypto::publicKey::D_MARKER =false [static]

D (private) boolean marker.

os::smart_ptr<**streamPackageFrame**> crypto::publicKey::fePackage [private]

@ brief Algorithm used for encryption

os::multiLock crypto::publicKey::keyLock [private]

@ brief Mutex for replacing the keys

os::smart_ptr<**number**> crypto::publicKey::n [protected]

@ brief Public key

const bool crypto::publicKey::N_MARKER =true [static]

N (public) boolean marker.

os::unsortedList<**number**> crypto::publicKey::oldD [protected]

@ brief List of old private keys

os::unsortedList<**number**> crypto::publicKey::oldN [protected]

@ brief List of old public keys

const bool crypto::publicKey::PRIVATE =false [static]

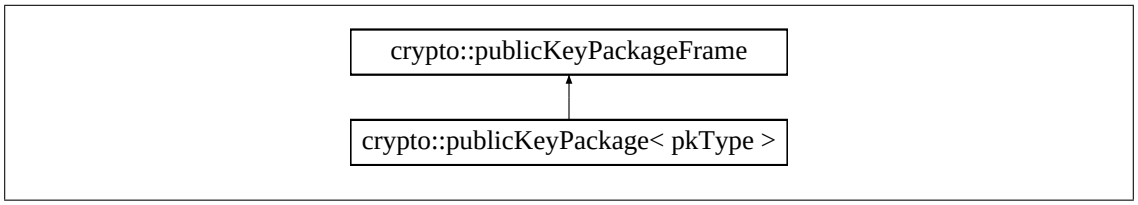
Private boolean marker.

const bool crypto::publicKey::PUBLIC =true [static]

Public boolean marker.

6.39 crypto::publicKeyPackage< pkType > Class Template Reference

Inheritance diagram for crypto::publicKeyPackage< pkType >:



Public Member Functions

- **publicKeyPackage** (uint16_t publicSize=size::public512)
- virtual ~**publicKeyPackage** ()
- os::smart_ptr< **publicKeyPackageFrame** > **getCopy** () const
- os::smart_ptr< **number** > **convert** (uint32_t *arr, uint16_t len) const
- os::smart_ptr< **number** > **convert** (const unsigned char *arr, unsigned int len) const
- os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const
- void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const
- void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const
- os::smart_ptr< **publicKey** > **generate** () const
- os::smart_ptr< **publicKey** > **bindKeys** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const
- os::smart_ptr< **publicKey** > **bindKeys** (uint32_t *_n, uint32_t *_d) const
- os::smart_ptr< **publicKey** > **openFile** (std::string fileName, std::string password) const
- os::smart_ptr< **publicKey** > **openFile** (std::string fileName, unsigned char *key, unsigned int keyLen) const
- std::string **algorithmName** () const
- uint16_t **algorithm** () const

Additional Inherited Members

6.39.1 Constructor & Destructor Documentation

template<class pkType > **crypto::publicKeyPackage**< pkType >::**publicKeyPackage** (uint16_t publicSize = `size::public512`) `[inline]`

template<class pkType > virtual **crypto::publicKeyPackage**< pkType >::~**publicKeyPackage** ( ) `[inline]`, `[virtual]`

6.39.2 Member Function Documentation

template<class pkType > uint16_t **crypto::publicKeyPackage**< pkType >::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 219).

template<class pkType > std::string **crypto::publicKeyPackage**< pkType >::algorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 219).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::bindKeys (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 219).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::bindKeys (uint32_t * _n, uint32_t * _d) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::convert (uint32_t * arr, uint16_t len) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::convert (const unsigned char * arr, unsigned int len) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > void **crypto::publicKeyPackage**< pkType >::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > void **crypto::publicKeyPackage**< pkType >::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::generate ( ) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**publicKeyPackageFrame**> **crypto::publicKeyPackage**< pkType >::getCopy ( ) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::openFile (std::string fileName, std::string password) const [inline], [virtual]

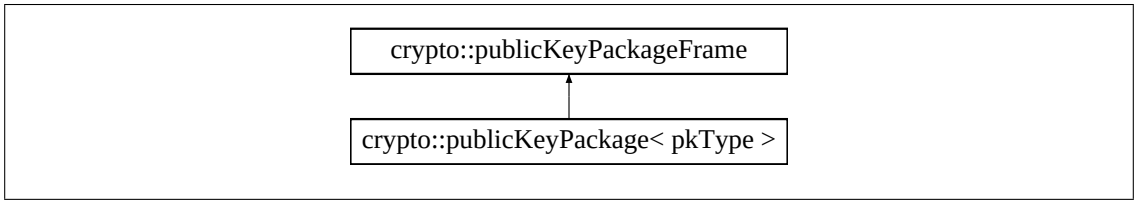
Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::openFile (std::string fileName, unsigned char * key, unsigned int keyLen) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 220).

6.40 crypto::publicKeyPackageFrame Class Reference

Inheritance diagram for crypto::publicKeyPackageFrame:



Public Member Functions

- **publicKeyPackageFrame** (uint16_t publicSize=size::public512)
- virtual ~**publicKeyPackageFrame** ()
- virtual os::smart_ptr< **publicKeyPackageFrame** > **getCopy** () const

- virtual os::smart_ptr< **number** > **convert** (uint32_t *arr, uint16_t len) const
- virtual os::smart_ptr< **number** > **convert** (const unsigned char *arr, unsigned int len) const
- virtual os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const
- virtual void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const
- virtual void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const
- virtual os::smart_ptr< **publicKey** > **generate** () const
- virtual os::smart_ptr< **publicKey** > **bindKeys** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const
- virtual os::smart_ptr< **publicKey** > **bindKeys** (uint32_t *_n, uint32_t *_d) const
- virtual os::smart_ptr< **publicKey** > **openFile** (std::string fileName, std::string password) const
- virtual os::smart_ptr< **publicKey** > **openFile** (std::string fileName, unsigned char *key, unsigned int keyLen) const
- virtual std::string **algorithmName** () const
- virtual uint16_t **algorithm** () const
- void **setKeySize** (uint16_t publicSize)
- uint16_t **keySize** () const

Protected Attributes

- uint16_t **_publicSize**

6.40.1 Constructor & Destructor Documentation

crypto::publicKeyPackageFrame::publicKeyPackageFrame (uint16_t publicSize = `size::public512`) `[inline]`

virtual crypto::publicKeyPackageFrame::~publicKeyPackageFrame ( ) `[inline]`, `[virtual]`

6.40.2 Member Function Documentation

virtual uint16_t crypto::publicKeyPackageFrame::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual std::string crypto::publicKeyPackageFrame::algorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::bindKeys (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const `[inline]`, `[virtual]`

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::bindKeys (uint32_t * _n, uint32_t * _d) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::convert (uint32_t * arr, uint16_t len) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::convert (const unsigned char * arr, unsigned int len) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 217).

virtual void crypto::publicKeyPackageFrame::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

virtual void crypto::publicKeyPackageFrame::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::generate ( ) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

virtual os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyPackageFrame::getCopy ( ) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

uint16_t crypto::publicKeyPackageFrame::keySize ( ) const [inline]

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::openFile (std::string fileName, std::string password) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::openFile (std::string fileName, unsigned char * key, unsigned int keyLen) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 218).

void crypto::publicKeyPackageFrame::setKeySize (uint16_t publicSize) [inline]

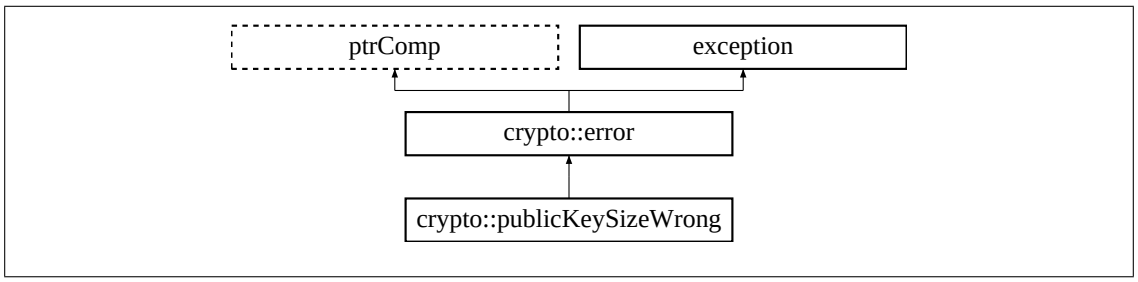
6.40.3 Member Data Documentation

uint16_t crypto::publicKeyPackageFrame::_publicSize [protected]

6.41 crypto::publicKeySizeWrong Class Reference

Public-key size error.

Inheritance diagram for crypto::publicKeySizeWrong:



Public Member Functions

- virtual ~**publicKeySizeWrong** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Public Key Size Wrong".
- std::string **errorDescription** () const

  Long error descriptor Returns "Attempted to use a code or n of improper size".

6.41.1 Detailed Description

Public-key size error.

Thrown when a public key or public key interaction detects a size mis-match or illegal size.

6.41.2 Constructor & Destructor Documentation

virtual crypto::publicKeySizeWrong::~publicKeySizeWrong ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.41.3 Member Function Documentation

std::string crypto::publicKeySizeWrong::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Attempted to use a code or n of improper size".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::publicKeySizeWrong::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Public Key Size Wrong".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.42 crypto::publicKeyTypeBank Class Reference

Public Member Functions

- virtual ~**publicKeyTypeBank** ()
- void **setDefaultPackage** (os::smart_ptr< **publicKeyPackageFrame** > package)
- const os::smart_ptr< **publicKeyPackageFrame** > **defaultPackage** () const
- void **pushPackage** (os::smart_ptr< **publicKeyPackageFrame** > package)
- const os::smart_ptr< **publicKeyPackageFrame** > **findPublicKey** (uint16_t pkID) const
- const os::smart_ptr< **publicKeyPackageFrame** > **findPublicKey** (const std::string &pkName) const

Static Public Member Functions

- static os::smart_ptr< **publicKeyTypeBank** > **singleton** ()

Private Member Functions

- **publicKeyTypeBank** ()

Private Attributes

- os::smart_ptr< **publicKeyPackageFrame** > **_defaultPackage**
- std::vector< os::smart_ptr< **publicKeyPackageFrame** > > **packageVector**

6.42.1 Constructor & Destructor Documentation

crypto::publicKeyTypeBank::publicKeyTypeBank ( ) [private]

virtual crypto::publicKeyTypeBank::~publicKeyTypeBank ( ) [inline], [virtual]

6.42.2 Member Function Documentation

const os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::defaultPackage ( ) const `[inline]`

const os::smart_ptr< **publicKeyPackageFrame** > crypto::publicKeyTypeBank::findPublicKey (uint16_t pkID) const

const os::smart_ptr< **publicKeyPackageFrame** > crypto::publicKeyTypeBank::findPublicKey (const std::string & pkName) const

void crypto::publicKeyTypeBank::pushPackage (os::smart_ptr< **publicKeyPackageFrame** > package)

void crypto::publicKeyTypeBank::setDefaultPackage (os::smart_ptr< **publicKeyPackageFrame** > package)

os::smart_ptr< **publicKeyTypeBank** > crypto::publicKeyTypeBank::singleton ( ) `[static]`

6.42.3 Member Data Documentation

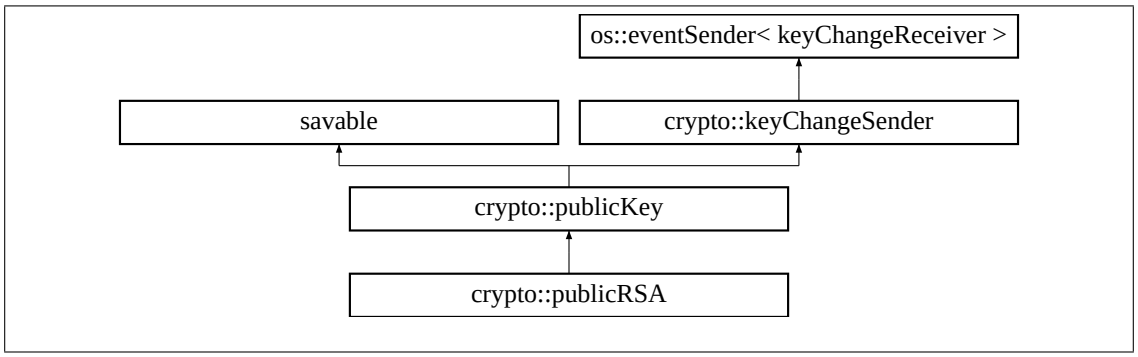
os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::_defaultPackage `[private]`

std::vector<os::smart_ptr<**publicKeyPackageFrame**> > crypto::publicKeyTypeBank::packageVector `[private]`

6.43 crypto::publicRSA Class Reference

RSA public-key encryption.

Inheritance diagram for crypto::publicRSA:



Public Member Functions

- **publicRSA** (uint16_t sz=size::public256)

  Default RSA constructor.
- **publicRSA** (**publicRSA** &ky)

  Copy Constructor.
- **publicRSA** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d, uint16_t sz=size::public512, uint64_t tms=os::getTimestamp())

  Construct with keys.

- **publicRSA** (uint32_t *_n, uint32_t *_d, uint16_t sz=size::public512, uint64_t tms=os::get↩ Timestamp())

  Construct with key arrays.
- **publicRSA** (std::string **fileName**, std::string password="", os::smart_ptr< **streamPackage**↩ **Frame** > stream_algo=NULL)

  Construct with path to file and password.
- **publicRSA** (std::string **fileName**, unsigned char *key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- virtual ~**publicRSA** ()

  Virtual destructor.
- os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num) const

  Converts number to integer.
- os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len) const

  Converts array to integer.
- os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len) const

  Converts byte array to integer.
- std::string **algorithmName** () const

  Access algorithm name.
- bool **generating** ()

  Tests if the keys are in the process of generating.
- void **generateNewKeys** ()

  Key generation function.
- os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN=NULL) const

  Number encode.
- void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN=N↩ ULL) const

  Hybrid data encode against number.
- void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const

  Data encode against number.
- os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code) const

  Number decode.
- os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code, unsigned int hist)

  Old number decode.

Static Public Member Functions

- static os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num, uint16_t **size**)

  Converts number to integer, statically.
- static os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len, uint16_t **size**)

  Converts array to integer, statically.

- static os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len, uint16_t **size**)

  Converts byte array to integer, statically.
- static uint16_t **staticAlgorithm** ()

  Access algorithm ID.
- static std::string **staticAlgorithmName** ()

  Access algorithm name.
- static os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Static number encode.
- static void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Static data encode.
- static void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength, uint16_t **size**)

  Static data encode.

Private Member Functions

- void **initE** ()

  *Subroutine initializing **crypto::publicRSA::e*** (p. *234*).

Private Attributes

- **integer e**

  Used in intermediate calculation.
- os::smart_ptr< **RSAKeyGenerator** > **keyGen**

  Key generation class.

Friends

- class **RSAKeyGenerator**

  Friendship with key generation.

Additional Inherited Members

6.43.1 Detailed Description

RSA public-key encryption.

This class defines an RSA algorithm for public-key cryptography.

6.43.2 Constructor & Destructor Documentation

crypto::publicRSA::publicRSA (uint16_t sz = `size::public256`)

Default RSA constructor.

Initializes and generates keys for a new pair of RSA keys. This serves as the default constructor for RSA keys.

Parameters

in	*sz*	Size of keys, crypto::size::public256 by default

crypto::publicRSA::publicRSA (**publicRSA** & ky)

Copy Constructor.

Copies the keys in one RSA pair into another. This copying includes all historical records as well.

Parameters

in	*ky*	Key pair to be copied

crypto::publicRSA::publicRSA (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`)

Construct with keys.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicRSA::publicRSA (uint32_t * _n, uint32_t * _d, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`)

Construct with key arrays.

Parameters

_n	Array of public key
_d	Array of private key
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicRSA::publicRSA (std::string fileName, std::string password = "", os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with path to file and password.

Parameters

fileName	Name of file to find keys
password	String representing symmetric key, "" by default
stream_algo	Symmetric key encryption algorithm, NULL by default

crypto::publicRSA::publicRSA (std::string fileName, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with path to file and password.

Parameters

fileName	Name of file to find keys
key	Symmetric key
keyLen	Length of symmetric key
stream_algo	Symmetric key encryption algorithm, NULL by default

virtual crypto::publicRSA::~publicRSA ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.43.3 Member Function Documentation

std::string crypto::publicRSA::algorithmName ( ) const `[inline]`, `[virtual]`

Access algorithm name.

Returns

crypto::publicRSA::staticAlgorithmName() (p. 234)

Reimplemented from **crypto::publicKey** (p. 202).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const os::smart_ptr< **number** > num) const `[virtual]`

Converts number to integer.

Parameters

`in`	*num*	Number to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 202).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const uint32_t * arr, uint16_t len) const [virtual]

Converts array to integer.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 203).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const unsigned char * arr, unsigned int len) const [virtual]

Converts byte array to integer.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 203).

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const os::smart_ptr< **number** > num, uint16_t size) [static]

Converts number to integer, statically.

Parameters

in	*num*	Number to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const uint32_t * arr, uint16_t len, uint16_t size) [static]

Converts array to integer, statically.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const unsigned char * arr, unsigned int len, uint16_t size) [static]

Converts byte array to integer, statically.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

os::smart_ptr<**number**> crypto::publicRSA::decode (os::smart_ptr< **number** > code) const [virtual]

Number decode.

Uses the private key to decode a set of data based on the RSA algorithm.

Parameters

in	*code*	Data to be decoded

Returns

Decoded number

Reimplemented from **crypto::publicKey** (p. 204).

os::smart_ptr<**number**> crypto::publicRSA::decode (os::smart_ptr< **number** > code, unsigned int hist) [virtual]

Old number decode.

Uses old private keys to decode a set of data based on the RSA algorithm.

Parameters

in	*code*	Data to be decoded
in	*hist*	Index of historical key

Returns

Decoded number

Reimplemented from **crypto::publicKey** (p. 204).

static os::smart_ptr<**number**> crypto::publicRSA::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static number encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

Encoded number

static void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static data encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

void

static void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength, uint16_t size) [static]

Static data encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*nLength*	Length of key array
in	*size*	Size of key used

Returns

void

os::smart_ptr<**number**> crypto::publicRSA::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN = NULL) const [virtual]

Number encode.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against, NULL by default

Returns

Encoded number

Reimplemented from **crypto::publicKey** (p. 207).

void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN = NULL) const [virtual]

Hybrid data encode against number.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented from **crypto::publicKey** (p. 207).

void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [virtual]

Data encode against number.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented from **crypto::publicKey** (p. 207).

void crypto::publicRSA::generateNewKeys ( ) [virtual]

Key generation function.
Generates new keys for the specific algorithm. This is re-implemented by every algorithm.

Returns

void

Reimplemented from **crypto::publicKey** (p. 208).

bool crypto::publicRSA::generating ( ) [virtual]

Tests if the keys are in the process of generating.

Returns

True if generating new keys

Reimplemented from **crypto::publicKey** (p. 208).

void crypto::publicRSA::initE ( ) [private]

Subroutine initializing **crypto::publicRSA::e** (p. 234).

static uint16_t crypto::publicRSA::staticAlgorithm ( ) [inline], [static]

Access algorithm ID.

Returns

crypto::algo::publicRSA

static std::string crypto::publicRSA::staticAlgorithmName ( ) [inline], [static]

Access algorithm name.

Returns

"RSA"

6.43.4 Friends And Related Function Documentation

friend class **RSAKeyGenerator** [friend]

Friendship with key generation.

The **crypto::RSAKeyGenerator** (p. 240) must be able to access the private members of the RSA public key class to bind newly generated keys.

6.43.5 Member Data Documentation

integer crypto::publicRSA::e [private]

Used in intermediate calculation.

os::smart_ptr<**RSAKeyGenerator**> crypto::publicRSA::keyGen [private]

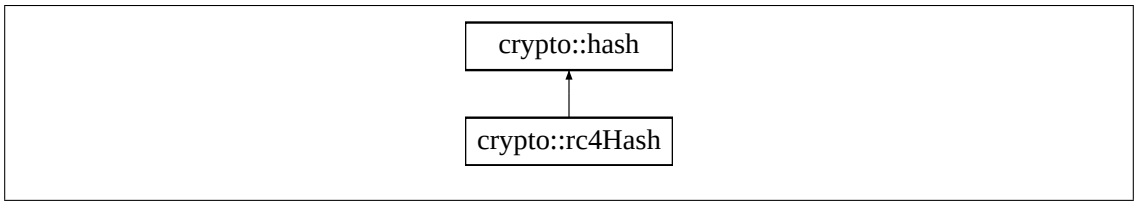
Key generation class.

This pointer will be NULL unless a key is currently being generated/

6.44 crypto::rc4Hash Class Reference

RC-4 hash class.

Inheritance diagram for crypto::rc4Hash:



Public Member Functions

- **rc4Hash** ()

  Default RC-4 hash constructor.
- **rc4Hash** (const unsigned char ∗**data**, uint16_t **size**)

  Raw data copy.
- **rc4Hash** (const **rc4Hash** &cpy)

  RC-4 copy constructor.
- void **preformHash** (const unsigned char ∗**data**, uint32_t dLen)

  Binds a data-set.
- std::string **algorithmName** () const

  Algorithm name string access.

Static Public Member Functions

- static std::string **staticAlgorithmName** ()

  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()

  Algorithm ID number access.
- static **rc4Hash hash64Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 64 bit hash.
- static **rc4Hash hash128Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 128 bit hash.
- static **rc4Hash hash256Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 256 bit hash.
- static **rc4Hash hash512Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 512 bit hash.

Private Member Functions

- **rc4Hash** (const unsigned char ∗**data**, uint32_t length, uint16_t **size**)

  RC-4 hash constructor.

Additional Inherited Members

6.44.1 Detailed Description

RC-4 hash class.

This class defines an RC-4 based hash. Note that this hash is likely cryptographically secure, but not proven cryptographically secure.

6.44.2 Constructor & Destructor Documentation

crypto::rc4Hash::rc4Hash (const unsigned char * data, uint32_t length, uint16_t size) [private]

RC-4 hash constructor.

Constructs a hash with the data to be hashed, the length of the array and the size of the hash to be constructed.

Parameters

in	*data*	Data array
in	*length*	Length of data array
in	*size*	Size of hash

crypto::rc4Hash::rc4Hash ( ) [inline]

Default RC-4 hash constructor.

Constructs an empty RC-4 hash class.

crypto::rc4Hash::rc4Hash (const unsigned char * data, uint16_t size)

Raw data copy.

Initializes the RC-4 hash with a data array. This data array is not hashed but assumed to represent hashed data.

Parameters

in	*data*	Hashed data array
in	*size*	Size of hash array

crypto::rc4Hash::rc4Hash (const **rc4Hash** & cpy) [inline]

RC-4 copy constructor.

Constructs an RC-4 hash with another RC-4 hash.

Parameters

in	*cpy*	Hash to be copied

6.44.3 Member Function Documentation

std::string crypto::rc4Hash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.

Returns the name of the current algorithm string. This function requires an instantiated RC-4 hash.

Returns

"RC-4"

Reimplemented from **crypto::hash** (p. 116).

static **rc4Hash** crypto::rc4Hash::hash128Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 128 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 128 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **rc4Hash** crypto::rc4Hash::hash256Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 256 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 256 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **rc4Hash** crypto::rc4Hash::hash512Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 512 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 512 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **rc4Hash** crypto::rc4Hash::hash64Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 64 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 64 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

void crypto::rc4Hash::preformHash (const unsigned char * data, uint32_t dLen)

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

static uint16_t crypto::rc4Hash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.

Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashRC4

static std::string crypto::rc4Hash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.

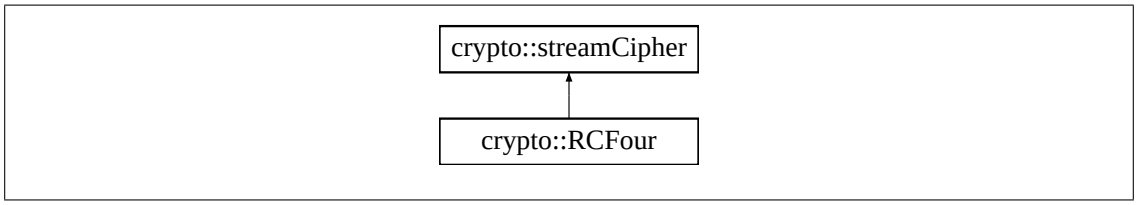
Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"RC-4"

6.45 crypto::RCFour Class Reference

Inheritance diagram for crypto::RCFour:



Public Member Functions

- **RCFour** (uint8_t *arr, int len)
- virtual ~**RCFour** ()
- uint8_t **getNext** ()
- uint16_t **algorithm** () const
- const std::string **algorithmName** () const

Static Public Member Functions

- static uint16_t **staticAlgorithm** ()
- static std::string **staticAlgorithmName** ()

Private Attributes

- uint8_t * **SArray**
- int **i**
- int **j**
- int **u**

6.45.1 Constructor & Destructor Documentation

RCFour::RCFour (uint8_t * arr, int len)

RCFour::~RCFour ( ) `[virtual]`

6.45.2 Member Function Documentation

uint16_t crypto::RCFour::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamCipher** (p. 242).

const std::string crypto::RCFour::algorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamCipher** (p. 242).

uint8_t RCFour::getNext ( ) [virtual]

Reimplemented from **crypto::streamCipher** (p. 242).

static uint16_t crypto::RCFour::staticAlgorithm ( ) [inline], [static]

static std::string crypto::RCFour::staticAlgorithmName ( ) [inline], [static]

6.45.3 Member Data Documentation

int crypto::RCFour::i [private]

int crypto::RCFour::j [private]

uint8_t* crypto::RCFour::SArray [private]

int crypto::RCFour::u [private]

6.46 crypto::RSAKeyGenerator Class Reference

Helper key generation class.

Public Member Functions

- **RSAKeyGenerator** (**publicRSA** &m)
  Constructs a generator with an RSA key.
- virtual ~**RSAKeyGenerator** ()
  Virtual destructor.
- **integer generatePrime** ()
  Generates a prime number.
- void **pushValues** ()
  Bind generated keys to master.

Public Attributes

- **integer p**
  Intermediate prime.
- **integer q**
  Intermediate prime.

Private Attributes

- **publicRSA** * **master**
  Pointer to keys.

6.46.1 Detailed Description

Helper key generation class.
This class helps to generate RSA keys. Once keys are generated, this class is destroyed.

6.46.2 Constructor & Destructor Documentation

crypto::RSAKeyGenerator::RSAKeyGenerator (**publicRSA** & m)

Constructs a generator with an RSA key.

This class is meaningless without a a reference to an RSA key to bind newly created keys to.

virtual crypto::RSAKeyGenerator::~RSAKeyGenerator ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.46.3 Member Function Documentation

integer crypto::RSAKeyGenerator::generatePrime ( )

Generates a prime number.

Returns

Prime integer

void crypto::RSAKeyGenerator::pushValues ( )

Bind generated keys to master.

Returns

void

6.46.4 Member Data Documentation

publicRSA∗ crypto::RSAKeyGenerator::master [private]

Pointer to keys.

Points to the RSA keys this generator will be placing its generated keys into.

integer crypto::RSAKeyGenerator::p

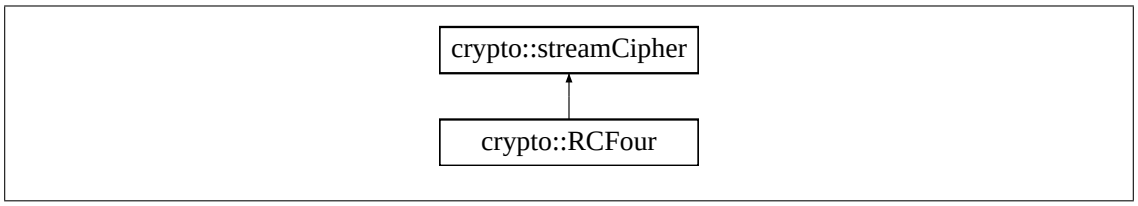
Intermediate prime.

integer crypto::RSAKeyGenerator::q

Intermediate prime.

6.47 crypto::streamCipher Class Reference

Inheritance diagram for crypto::streamCipher:



Public Member Functions

- virtual ~**streamCipher** ()
- virtual uint8_t **getNext** ()
- virtual uint16_t **algorithm** () const
- virtual const std::string **algorithmName** () const

Static Public Member Functions

- static uint16_t **staticAlgorithm** ()
- static std::string **staticAlgorithmName** ()

6.47.1 Constructor & Destructor Documentation

virtual crypto::streamCipher::~streamCipher ( ) `[inline]`, `[virtual]`

6.47.2 Member Function Documentation

virtual uint16_t crypto::streamCipher::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::RCFour** (p. 239).

virtual const std::string crypto::streamCipher::algorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::RCFour** (p. 239).

virtual uint8_t crypto::streamCipher::getNext ( ) `[inline]`, `[virtual]`

Reimplemented in **crypto::RCFour** (p. 240).

static uint16_t crypto::streamCipher::staticAlgorithm ( ) `[inline]`, `[static]`

static std::string crypto::streamCipher::staticAlgorithmName ( ) `[inline]`, `[static]`

6.48 crypto::streamDecrypter Class Reference

Public Member Functions

- **streamDecrypter** (os::smart_ptr< **streamCipher** > c)

- virtual ~**streamDecrypter** ()
- uint8_t * **recieveData** (uint8_t *array, unsigned int len, uint16_t flag)

Private Attributes

- os::smart_ptr< **streamCipher** > **cipher**
- **streamPacket** ** **packetArray**
- unsigned int **last_value**
- unsigned int **mid_value**

6.48.1 Constructor & Destructor Documentation

streamDecrypter::streamDecrypter (os::smart_ptr< **streamCipher** > c)

streamDecrypter::~streamDecrypter ( ) [virtual]

6.48.2 Member Function Documentation

uint8_t * streamDecrypter::recieveData (uint8_t * array, unsigned int len, uint16_t flag)

6.48.3 Member Data Documentation

os::smart_ptr<**streamCipher**> crypto::streamDecrypter::cipher [private]

unsigned int crypto::streamDecrypter::last_value [private]

unsigned int crypto::streamDecrypter::mid_value [private]

streamPacket ** crypto::streamDecrypter::packetArray [private]

6.49 crypto::streamEncrypter Class Reference

Public Member Functions

- **streamEncrypter** (os::smart_ptr< **streamCipher** > c)
- virtual ~**streamEncrypter** ()
- uint8_t * **sendData** (uint8_t *array, unsigned int len, uint16_t &flag)

Private Attributes

- os::smart_ptr< **streamCipher** > **cipher**
- unsigned int **last_loc**
- uint16_t * **ID_check**

6.49.1 Constructor & Destructor Documentation

streamEncrypter::streamEncrypter (os::smart_ptr< **streamCipher** > c)

streamEncrypter::~streamEncrypter ( ) [virtual]

6.49.2 Member Function Documentation

uint8_t * streamEncrypter::sendData (uint8_t * array, unsigned int len, uint16_t & flag)

6.49.3 Member Data Documentation

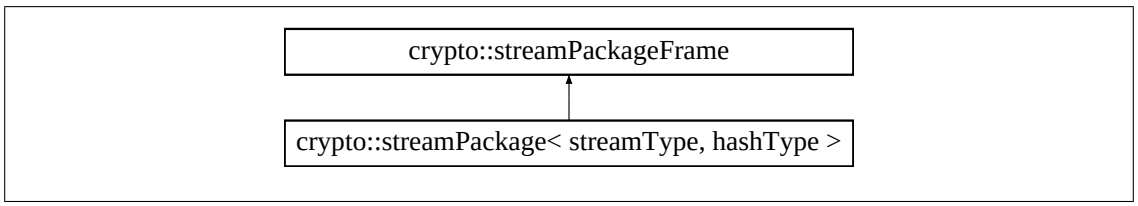
os::smart_ptr<**streamCipher**> crypto::streamEncrypter::cipher [private]

uint16_t* crypto::streamEncrypter::ID_check [private]

unsigned int crypto::streamEncrypter::last_loc [private]

6.50 crypto::streamPackage< streamType, hashType > Class Template Reference

Inheritance diagram for crypto::streamPackage< streamType, hashType >:



Public Member Functions

- **streamPackage** (uint16_t **hashSize**=size::hash256)
- virtual ~**streamPackage** ()
- os::smart_ptr< **streamPackageFrame** > **getCopy** () const
- **hash hashEmpty** () const
- **hash hashData** (unsigned char *data, uint32_t len) const
- **hash hashCopy** (unsigned char *data) const
- os::smart_ptr< **streamCipher** > **buildStream** (unsigned char *data, uint32_t len) const
- std::string **streamAlgorithmName** () const
- uint16_t **streamAlgorithm** () const
- std::string **hashAlgorithmName** () const
- uint16_t **hashAlgorithm** () const

Additional Inherited Members

6.50.1 Constructor & Destructor Documentation

template<class streamType , class hashType > **crypto::streamPackage**< streamType, hashType >::**streamPackage** (uint16_t hashSize = size::hash256) [inline]

template<class streamType , class hashType > virtual **crypto::streamPackage**< streamType, hashType >::~**streamPackage** ( ) [inline], [virtual]

6.50.2 Member Function Documentation

template<class streamType , class hashType > os::smart_ptr<**streamCipher**>
crypto::streamPackage< streamType, hashType >::buildStream (unsigned char * data, uint32_t len) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 246).

template<class streamType , class hashType > os::smart_ptr<**streamPackageFrame**>
crypto::streamPackage< streamType, hashType >::getCopy ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 246).

template<class streamType , class hashType > uint16_t **crypto::streamPackage**< streamType, hashType >::hashAlgorithm ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > std::string **crypto::streamPackage**< streamType, hashType >::hashAlgorithmName ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashCopy (unsigned char * data) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashData (unsigned char * data, uint32_t len) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashEmpty ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > uint16_t **crypto::streamPackage**< streamType, hashType >::streamAlgorithm ( ) const [inline], [virtual]

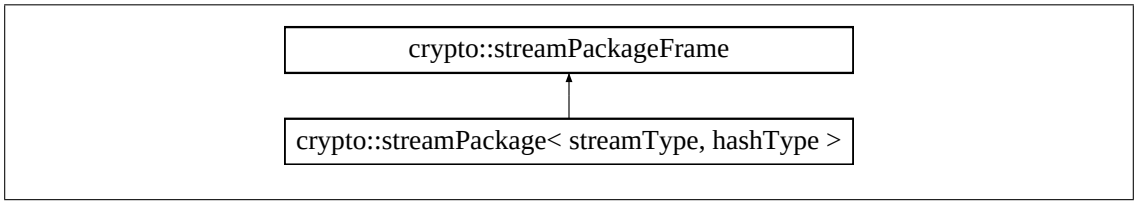
Reimplemented from **crypto::streamPackageFrame** (p. 247).

template<class streamType , class hashType > std::string **crypto::streamPackage**< streamType, hashType >::streamAlgorithmName ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 247).

6.51 crypto::streamPackageFrame Class Reference

Inheritance diagram for crypto::streamPackageFrame:



Public Member Functions

- **streamPackageFrame** (uint16_t **hashSize**=size::hash256)
- virtual ~**streamPackageFrame** ()
- virtual os::smart_ptr< **streamPackageFrame** > **getCopy** () const
- virtual **hash hashEmpty** () const
- virtual **hash hashData** (unsigned char *data, uint32_t len) const
- virtual **hash hashCopy** (unsigned char *data) const
- virtual os::smart_ptr< **streamCipher** > **buildStream** (unsigned char *data, uint32_t len) const
- virtual std::string **streamAlgorithmName** () const
- virtual uint16_t **streamAlgorithm** () const
- virtual std::string **hashAlgorithmName** () const
- virtual uint16_t **hashAlgorithm** () const
- void **setHashSize** (uint16_t **hashSize**)
- uint16_t **hashSize** () const

Protected Attributes

- uint16_t **_hashSize**

6.51.1 Constructor & Destructor Documentation

crypto::streamPackageFrame::streamPackageFrame (uint16_t hashSize = `size::hash256`) `[inline]`

virtual crypto::streamPackageFrame::~streamPackageFrame ( ) `[inline]`, `[virtual]`

6.51.2 Member Function Documentation

virtual os::smart_ptr<**streamCipher**> crypto::streamPackageFrame::buildStream (unsigned char * data, uint32_t len) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual os::smart_ptr<**streamPackageFrame**> crypto::streamPackageFrame::getCopy ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual uint16_t crypto::streamPackageFrame::hashAlgorithm ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual std::string crypto::streamPackageFrame::hashAlgorithmName ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual **hash** crypto::streamPackageFrame::hashCopy (unsigned char * data) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual **hash** crypto::streamPackageFrame::hashData (unsigned char * data, uint32_t len) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual **hash** crypto::streamPackageFrame::hashEmpty ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

uint16_t crypto::streamPackageFrame::hashSize ( ) const [inline]

void crypto::streamPackageFrame::setHashSize (uint16_t hashSize) [inline]

virtual uint16_t crypto::streamPackageFrame::streamAlgorithm ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

virtual std::string crypto::streamPackageFrame::streamAlgorithmName ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 245).

6.51.3 Member Data Documentation

uint16_t crypto::streamPackageFrame::_hashSize [protected]

6.52 crypto::streamPackageTypeBank Class Reference

Public Member Functions

- virtual ~**streamPackageTypeBank** ()
- void **setDefaultPackage** (os::smart_ptr< **streamPackageFrame** > package)
- const os::smart_ptr< **streamPackageFrame** > **defaultPackage** () const
- void **pushPackage** (os::smart_ptr< **streamPackageFrame** > package)
- const os::smart_ptr< **streamPackageFrame** > **findStream** (uint16_t streamID, uint16_t hash↩ ID) const

- const os::smart_ptr< **streamPackageFrame** > **findStream** (const std::string &streamName, const std::string &hashName) const

Static Public Member Functions

- static os::smart_ptr< **streamPackageTypeBank** > **singleton** ()

Private Member Functions

- **streamPackageTypeBank** ()

Private Attributes

- os::smart_ptr< **streamPackageFrame** > **_defaultPackage**
- std::vector< os::smart_ptr< std::vector< os::smart_ptr< **streamPackageFrame** > > > > **packageVector**

6.52.1 Constructor & Destructor Documentation

crypto::streamPackageTypeBank::streamPackageTypeBank ( ) [private]

virtual crypto::streamPackageTypeBank::~streamPackageTypeBank ( ) [inline], [virtual]

6.52.2 Member Function Documentation

const os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::defaultPackage ( ) const [inline]

const os::smart_ptr< **streamPackageFrame** > crypto::streamPackageTypeBank::findStream (uint16_t streamID, uint16_t hashID) const

const os::smart_ptr< **streamPackageFrame** > crypto::streamPackageTypeBank::findStream (const std::string & streamName, const std::string & hashName) const

void crypto::streamPackageTypeBank::pushPackage (os::smart_ptr< **streamPackageFrame** > package)

void crypto::streamPackageTypeBank::setDefaultPackage (os::smart_ptr< **streamPackageFrame** > package)

os::smart_ptr< **streamPackageTypeBank** > crypto::streamPackageTypeBank::singleton ( ) [static]

6.52.3 Member Data Documentation

os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::_defaultPackage [private]

std::vector<os::smart_ptr<std::vector<os::smart_ptr<**streamPackageFrame**> > > > crypto::streamPackageTypeBank::packageVector [private]

6.53 crypto::streamPacket Class Reference

Public Member Functions

- **streamPacket** (os::smart_ptr< **streamCipher** > source, unsigned int s)
- virtual ~**streamPacket** ()
- uint16_t **getIdentifier** () const
- const uint8_t * **getPacket** () const
- uint8_t * **encrypt** (uint8_t *pt, unsigned int len, bool surpress=true) const

Private Attributes

- uint8_t * **packetArray**
- uint16_t **identifier**
- unsigned int **size**

6.53.1 Constructor & Destructor Documentation

streamPacket::streamPacket (os::smart_ptr< **streamCipher** > source, unsigned int s)

streamPacket::~streamPacket ( ) [virtual]

6.53.2 Member Function Documentation

uint8_t * streamPacket::encrypt (uint8_t * pt, unsigned int len, bool surpress = true) const

uint16_t streamPacket::getIdentifier ( ) const

const uint8_t * streamPacket::getPacket ( ) const

6.53.3 Member Data Documentation

uint16_t crypto::streamPacket::identifier [private]

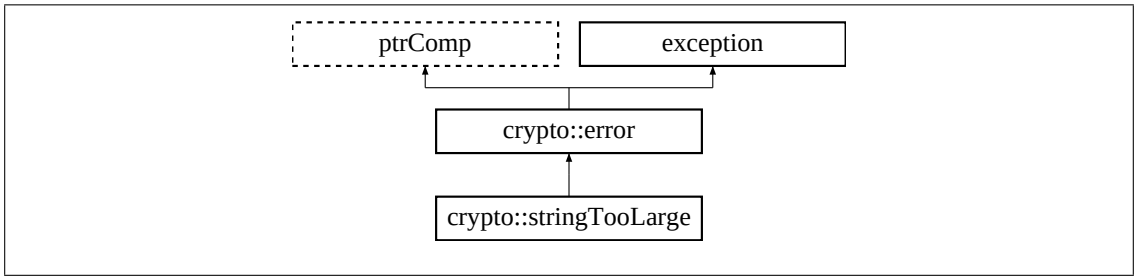
uint8_t* crypto::streamPacket::packetArray [private]

unsigned int crypto::streamPacket::size [private]

6.54 crypto::stringTooLarge Class Reference

String size error.

Inheritance diagram for crypto::stringTooLarge:



Public Member Functions

- virtual ~**stringTooLarge** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Group ID/Name Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Group ID or Name was larger than the maximum size. Please user a smaller string".

6.54.1 Detailed Description

String size error.

Thrown when either the username or group ID are too large.

6.54.2 Constructor & Destructor Documentation

virtual crypto::stringTooLarge::~stringTooLarge ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.54.3 Member Function Documentation

std::string crypto::stringTooLarge::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Group ID or Name was larger than the maximum size. Please user a smaller string".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::stringTooLarge::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Group ID/Name Size Error".

Returns

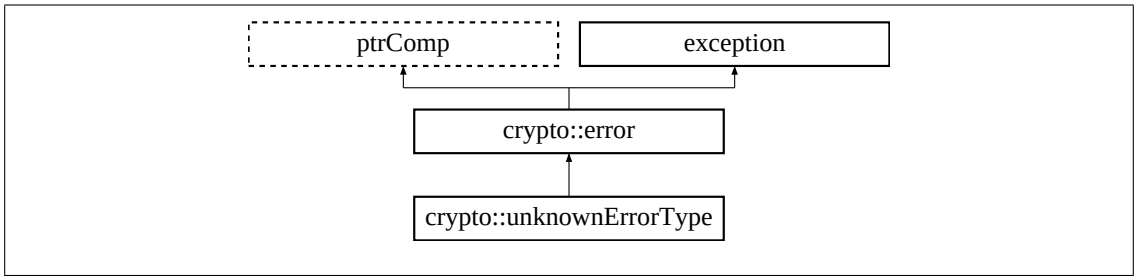
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.55 crypto::unknownErrorType Class Reference

Unknown error.

Inheritance diagram for crypto::unknownErrorType:



Public Member Functions

- virtual ~**unknownErrorType** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Unknown Error Type".
- std::string **errorDescription** () const

  Long error descriptor Returns "Caught some exception, but the type is unknown".

6.55.1 Detailed Description

Unknown error.

Thrown when an error of undefined type occurs. Used as a catch-all exception.

6.55.2 Constructor & Destructor Documentation

virtual crypto::unknownErrorType::~unknownErrorType ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.55.3 Member Function Documentation

std::string crypto::unknownErrorType::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Caught some exception, but the type is unknown".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 81).

std::string crypto::unknownErrorType::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Unknown Error Type".

Returns

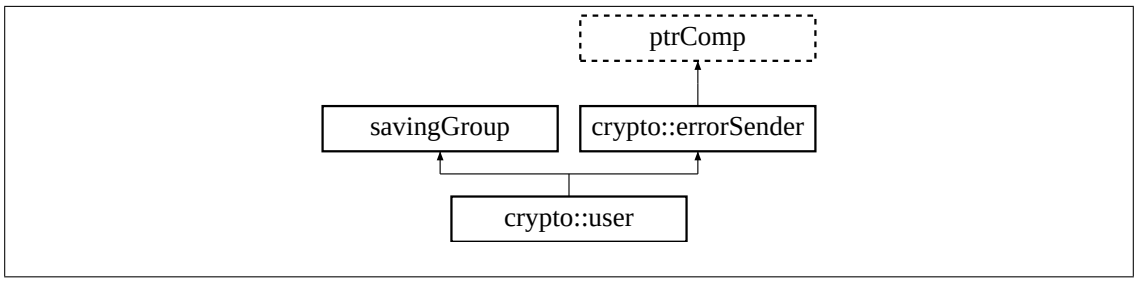
Error title std::string

Reimplemented from **crypto::error** (p. 82).

6.56 crypto::user Class Reference

Primary user class.

Inheritance diagram for crypto::user:



Public Member Functions

- bool **wasConstructed** () const

  Returns the construction state of the user.
- **user** (std::string **username**, std::string saveDir="", const unsigned char *key=NULL, unsigned int keyLen=0)

  Constructs the user from scratch or directory.
- virtual ~**user** ()

  Virtual destructor.
- void **save** ()

  Saves all dependencies.
- void **setPassword** (const unsigned char *key=NULL, unsigned int keyLen=0)

  Set password.
- void **setStreamPackage** (os::smart_ptr< **streamPackageFrame** > strmPack)

  Set stream package.
- bool **setDefaultPublicKey** (os::smart_ptr< **publicKey** > key)

  Sets the default public key.
- bool **addPublicKey** (os::smart_ptr< **publicKey** > key)

  Attempt to add new public key.
- os::smart_ptr< **publicKey** > **findPublicKey** (os::smart_ptr< **publicKeyPackageFrame** > pkfrm)

  Find public key by information.

- const std::string & **username** () const

  Access name of user.
- const unsigned char * **password** () const

  Access raw password.
- unsigned int **passwordLength** () const

  Access password length.
- std::string **directory** () const

  Access save directory.
- os::smart_ptr< **streamPackageFrame** > **streamPackage** () const

  Access streaming package.
- os::smart_ptr< **keyBank** > **getKeyBank** ()

  Access key bank.
- os::smart_ptr< **publicKey** > **getDefaultPublicKey** ()

  Returns the default public key.
- os::smart_ptr< os::adnode< **publicKey** > > **getFirstPublicKey** ()

  Returns the first public key group.
- os::smart_ptr< os::adnode< **publicKey** > > **getLastPublicKey** ()

  Returns the last public key group.
- os::smart_ptr< **gatewaySettings** > **findSettings** (std::string group="default")

  Find gateway settings.
- os::smart_ptr< **gatewaySettings** > **insertSettings** (std::string group)

  Insert gateway settings.
- os::smart_ptr< os::adnode< **gatewaySettings** > > **getFirstSettings** ()

  Returns the first gateway settings group.
- os::smart_ptr< os::adnode< **gatewaySettings** > > **getLastSettings** ()

  Returns the last gateway settings group.
- os::smart_ptr< **publicKey** > **searchKey** (**hash** hsh, unsigned int &hist, bool &type)

  Searches for key by hash.
- os::smart_ptr< **publicKey** > **searchKey** (os::smart_ptr< **number** > key, unsigned int &hist, bool &type)

  Searches for key.
- os::smart_ptr< **publicKey** > **searchKey** (**hash** hsh)

  Searches for key.
- os::smart_ptr< **publicKey** > **searchKey** (os::smart_ptr< **number** > key)

  Searches for key.

Protected Member Functions

- os::smartXMLNode **generateSaveTree** ()

  Creates meta-data XML file.

Protected Attributes

- bool **_wasConstructed**
- std::string **_username**

  Name of user.
- unsigned char * **_password**

  Primary symmetric key.
- unsigned int **_passwordLength**

  Length of symmetric key.
- std::string **_saveDir**

  Save directory for user.
- os::smart_ptr< **streamPackageFrame** > **_streamPackage**

  Default stream package.
- os::smart_ptr< **keyBank** > **_keyBank**

  Key bank.
- os::asyncAVLTree< **publicKey** > **_publicKeys**

  Public keys.
- os::smart_ptr< **publicKey** > **_defaultKey**

  Default public key.
- os::asyncAVLTree< **gatewaySettings** > **_settings**

  List of gateway settings.

6.56.1 Detailed Description

Primary user class.

The user class defines a set of keys associated with a local user. This class notifies a set of listeners when various passwords and keys are changed, as this class allows for the encryption of a group of files with the provided keys

6.56.2 Constructor & Destructor Documentation

crypto::user::user (std::string username, std::string saveDir = "", const unsigned char * key = NULL, unsigned int keyLen = 0)

Constructs the user from scratch or directory.

Constructs a user from a directory or from scratch. If the specified directory does not exists, this class creates the directory and begins to populate it. If no key is specified, all files are un-encrypted. If a key is specified, all files are encrypted with this key.

Parameters

in	*username*	Name of user to be saved
in	*saveDir*	Directory to save users in
in	*key*	Symetric key
in	*keyLen*	Length of symetric key

virtual crypto::user::~user ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.56.3 Member Function Documentation

bool crypto::user::addPublicKey (os::smart_ptr< **publicKey** > key)

Attempt to add new public key.

Attempts to add a public key to the public key bank. If successful, and if the default key is NULL, the added key becomes the default key.

Parameters

in	*key*	Public key to be added

Returns

True if successfully added, else, false

std::string crypto::user::directory ( ) const [inline]

Access save directory.

Returns

crypto::user::_saveDir (p. 261) + username

os::smart_ptr<**publicKey**> crypto::user::findPublicKey (os::smart_ptr< **publicKeyPackageFrame** > pkfrm)

Find public key by information.

Searches for a public key with the given' characteristics. Keys are searched by algorithm and size.

Parameters

in	*pkfrm*	Public key information to match

Returns

Public key matching intrinsics

os::smart_ptr<**gatewaySettings**> crypto::user::findSettings (std::string group = "default")

Find gateway settings.

Parameters

in	*group*	Name of group of the settings

Returns

Pointer to the found gateway settings

os::smartXMLNode crypto::user::generateSaveTree ( ) [protected]

Creates meta-data XML file.
Constructs and returns the XML tree for this class. The XML tree may or may not be encrypted.

Returns

XML tree for saving

os::smart_ptr<**publicKey**> crypto::user::getDefaultPublicKey ( ) [inline]

Returns the default public key.

Returns

crypto::user::_defaultKey (p. 260)

os::smart_ptr<os::adnode<**publicKey**> > crypto::user::getFirstPublicKey ( ) [inline]

Returns the first public key group.
Allows programs to list off the available key groups bound to this user

Returns

crypto::user::_publicKeys.getFirst()

os::smart_ptr<os::adnode<**gatewaySettings**> > crypto::user::getFirstSettings ( ) [inline]

Returns the first gateway settings group.
Allows programs to list off the available gateway settings bound to this user

Returns

crypto::user::_settings.getFirst()

os::smart_ptr<**keyBank**> crypto::user::getKeyBank ( ) [inline]

Access key bank.

Returns

crypto::user::_keyBank (p. 260)

os::smart_ptr<os::adnode<**publicKey**> > crypto::user::getLastPublicKey ( ) [inline]

Returns the last public key group.
Allows programs to list off the available key groups bound to this user

Returns

crypto::user::_publicKeys.getFirst()

os::smart_ptr<os::adnode<**gatewaySettings**> > crypto::user::getLastSettings ( ) [inline]

Returns the last gateway settings group.
Allows programs to list off the available gateway settings bound to this user

Returns

crypto::user::_settings.getLast()

os::smart_ptr<**gatewaySettings**> crypto::user::insertSettings (std::string group)

Insert gateway settings.

Parameters

in	*group*	Name of group of the settings

Returns

Point to the inserted gateway settings

const unsigned char* crypto::user::password ( ) const [inline]

Access raw password.

Returns

crypto::user::_password (p. 260)

unsigned int crypto::user::passwordLength ( ) const [inline]

Access password length.

Returns

crypto::user::_passwordLength (p. 260)

void crypto::user::save ( )

Saves all dependencies.
This function saves all dependencies based on the save queue.

Returns

void

os::smart_ptr<**publicKey**> crypto::user::searchKey (**hash** hsh, unsigned int & hist, bool & type)

Searches for key by hash.
Binds the location that the keys were found in to the arguments of the function.

Parameters

`in`	*hsh*	Hash of the key to be searched for
`out`	*hist*	History value the key was found
`out`	*type*	Type (public or private)

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (os::smart_ptr< **number** > key, unsigned int & hist, bool & type)

Searches for key.
Binds the location that the keys were found in to the arguments of the function.

Parameters

`in`	*num*	Key to search for
`out`	*hist*	History value the key was found
`out`	*type*	Type (public or private)

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (**hash** hsh) `[inline]`

Searches for key.

Parameters

`in`	*num*	Key to search for

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (os::smart_ptr< **number** > key) `[inline]`

Searches for key.

Parameters

in	*num*	Key to search for

Returns

Key pair conatining the searched key

bool crypto::user::setDefaultPublicKey (os::smart_ptr< **publicKey** > key)

Sets the default public key.

Attempts to bind a public key as the default public key. First checks if the key in question exists and binds the key with the characteristics of the provided key as the default key.

Parameters

in	*key*	Public key to be bound as the default key

Returns

True if default key bound, else, false

void crypto::user::setPassword (const unsigned char * key = NULL, unsigned int keyLen = 0)

Set password.

Sets symetric key used to securely save user data.

Parameters

in	*key*	Symetric key
in	*keyLen*	Length of symetric key

Returns

void

void crypto::user::setStreamPackage (os::smart_ptr< **streamPackageFrame** > strmPack)

Set stream package.

Binds a new stream package. Calls for saving of this user.

Parameters

in	*strmPack*	Stream package

Returns

void

os::smart_ptr<**streamPackageFrame**> crypto::user::streamPackage ( ) const [inline]

Access streaming package.

Returns

crypto::user::_streamPackage (p. 261)

const std::string& crypto::user::username ( ) const [inline]

Access name of user.

Returns

crypto::user::_username (p. 261)

bool crypto::user::wasConstructed ( ) const [inline]

Returns the construction state of the user.

Returns

crypto::bool::_wasConstructed

6.56.4 Member Data Documentation

os::smart_ptr<**publicKey**> crypto::user::_defaultKey [protected]

Default public key.

Sets the default public key definition. Note that a default public key will be defined the moment any public key is bound to a user.

os::smart_ptr<**keyBank**> crypto::user::_keyBank [protected]

Key bank.

This key bank defines all of the public keys which are known by this user

unsigned char* crypto::user::_password [protected]

Primary symmetric key.

unsigned int crypto::user::_passwordLength [protected]

Length of symmetric key.

os::asyncAVLTree<**publicKey**> crypto::user::_publicKeys [protected]

Public keys.

This stores all public keys accociated with this specific user.

std::string crypto::user::_saveDir [protected]

Save directory for user.

os::asyncAVLTree<**gatewaySettings**> crypto::user::_settings [protected]

List of gateway settings.

os::smart_ptr<**streamPackageFrame**> crypto::user::_streamPackage [protected]

Default stream package.

std::string crypto::user::_username [protected]

Name of user.

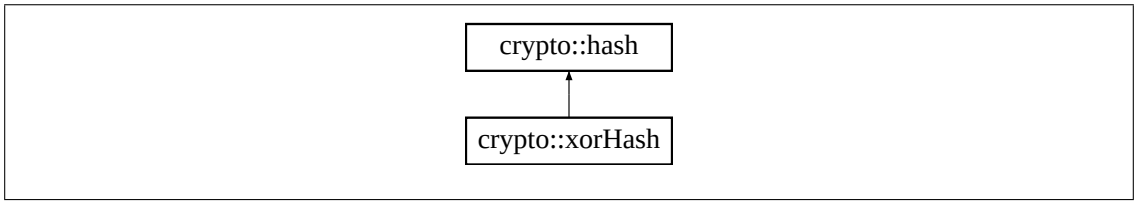
bool crypto::user::_wasConstructed [protected]

Stores if the user was constructed

6.57 crypto::xorHash Class Reference

XOR hash class.

Inheritance diagram for crypto::xorHash:



Public Member Functions

- **xorHash** ()

  Default XOR hash constructor.
- **xorHash** (const unsigned char ∗**data**, uint16_t **size**)

  Raw data copy.
- **xorHash** (const **xorHash** &cpy)

  XOR copy constructor.
- void **preformHash** (const unsigned char ∗**data**, uint32_t dLen)

  Binds a data-set.
- std::string **algorithmName** () const

  Algorithm name string access.

Static Public Member Functions

- static std::string **staticAlgorithmName** ()

  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()

  Algorithm ID number access.
- static **xorHash hash64Bit** (const unsigned char *∗**data**, uint32_t length)

  Static 64 bit hash.
- static **xorHash hash128Bit** (const unsigned char *∗**data**, uint32_t length)

  Static 128 bit hash.
- static **xorHash hash256Bit** (const unsigned char *∗**data**, uint32_t length)

  Static 256 bit hash.
- static **xorHash hash512Bit** (const unsigned char *∗**data**, uint32_t length)

  Static 512 bit hash.

Private Member Functions

- **xorHash** (const unsigned char *∗**data**, uint32_t length, uint16_t **size**)

  XOR hash constructor.

Additional Inherited Members

6.57.1 Detailed Description

XOR hash class.

This class defines an XOR based hash. Note that this hash is not cryptographically secure and essentially just acts as a checksum.

6.57.2 Constructor & Destructor Documentation

crypto::xorHash::xorHash (const unsigned char * data, uint32_t length, uint16_t size) `[private]`

XOR hash constructor.

Constructs a hash with the data to be hashed, the length of the array and the size of the hash to be constructed.

Parameters

`in`	*data*	Data array
`in`	*length*	Length of data array
`in`	*size*	Size of hash

crypto::xorHash::xorHash ( ) `[inline]`

Default XOR hash constructor.

Constructs an empty XOR hash class.

crypto::xorHash::xorHash (const unsigned char * data, uint16_t size)

Raw data copy.

Initializes the XOR hash with a data array. This data array is not hashed but assumed to represent hashed data.

Parameters

in	*data*	Hashed data array
in	*size*	Size of hash array

crypto::xorHash::xorHash (const **xorHash** & cpy) [inline]

XOR copy constructor.

Constructs an XOR hash with another XOR hash.

Parameters

in	*cpy*	Hash to be copied

6.57.3 Member Function Documentation

std::string crypto::xorHash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.

Returns the name of the current algorithm string. This function requires an instantiated XOR hash.

Returns

"XOR"

Reimplemented from **crypto::hash** (p. 116).

static **xorHash** crypto::xorHash::hash128Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 128 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 128 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **xorHash** crypto::xorHash::hash256Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 256 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 256 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **xorHash** crypto::xorHash::hash512Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 512 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 512 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

static **xorHash** crypto::xorHash::hash64Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 64 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 64 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 261)

void crypto::xorHash::preformHash (const unsigned char * data, uint32_t dLen)

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

static uint16_t crypto::xorHash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.

Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashXOR

static std::string crypto::xorHash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.

Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"XOR"