

Qt Cryptographic Architecture Reference Manual

Generated by Doxygen 1.4.6

Fri Jul 6 13:22:42 2007

Contents

1	Qt Cryptographic Architecture	1
1.1	Features	1
1.2	Using QCA	2
1.3	Availability	3
2	Qt Cryptographic Architecture Directory Hierarchy	5
2.1	Qt Cryptographic Architecture Directories	5
3	Qt Cryptographic Architecture Namespace Index	7
3.1	Qt Cryptographic Architecture Namespace List	7
4	Qt Cryptographic Architecture Hierarchical Index	9
4.1	Qt Cryptographic Architecture Class Hierarchy	9
5	Qt Cryptographic Architecture Class Index	13
5.1	Qt Cryptographic Architecture Class List	13
6	Qt Cryptographic Architecture File Index	15
6.1	Qt Cryptographic Architecture File List	15
7	Qt Cryptographic Architecture Page Index	17
7.1	Qt Cryptographic Architecture Related Pages	17
8	Qt Cryptographic Architecture Directory Documentation	19
8.1	examples/ Directory Reference	19
8.2	include/ Directory Reference	20
8.3	include/QtCrypto/ Directory Reference	21
9	Qt Cryptographic Architecture Namespace Documentation	23
9.1	QCA Namespace Reference	23
10	Qt Cryptographic Architecture Class Documentation	47

10.1 QCA::AbstractLogDevice Class Reference	47
10.2 QCA::Algorithm Class Reference	49
10.3 QCA::Base64 Class Reference	52
10.4 QCA::BasicContext Class Reference	55
10.5 QCA::BigInteger Class Reference	57
10.6 QCA::BufferedComputation Class Reference	63
10.7 QCA::Certificate Class Reference	65
10.8 QCA::CertificateAuthority Class Reference	74
10.9 QCA::CertificateChain Class Reference	77
10.10 QCA::CertificateCollection Class Reference	80
10.11 QCA::CertificateInfoOrdered Class Reference	84
10.12 QCA::CertificateInfoPair Class Reference	85
10.13 QCA::CertificateInfoType Class Reference	87
10.14 QCA::CertificateOptions Class Reference	90
10.15 QCA::CertificateRequest Class Reference	96
10.16 QCA::Cipher Class Reference	103
10.17 QCA::CMS Class Reference	108
10.18 QCA::ConstraintType Class Reference	111
10.19 QCA::CRL Class Reference	114
10.20 QCA::CRLEntry Class Reference	119
10.21 QCA::DHPrivateKey Class Reference	122
10.22 QCA::DHPublicKey Class Reference	124
10.23 QCA::DLGroup Class Reference	126
10.24 QCA::DSAPrivateKey Class Reference	129
10.25 QCA::DSAPublicKey Class Reference	131
10.26 QCA::Event Class Reference	133
10.27 QCA::EventHandler Class Reference	138
10.28 QCA::FileWatch Class Reference	141
10.29 QCA::Filter Class Reference	143
10.30 QCA::Hash Class Reference	145
10.31 QCA::Hex Class Reference	150
10.32 QCA::InitializationVector Class Reference	152
10.33 QCA::Initializer Class Reference	154
10.34 QCA::KeyBundle Class Reference	155
10.35 QCA::KeyDerivationFunction Class Reference	161
10.36 QCA::KeyGenerator Class Reference	163

10.37QCA::KeyLength Class Reference	167
10.38QCA::KeyLoader Class Reference	169
10.39QCA::KeyStore Class Reference	172
10.40QCA::KeyStoreEntry Class Reference	177
10.41QCA::KeyStoreEntryWatcher Class Reference	183
10.42QCA::KeyStoreInfo Class Reference	185
10.43QCA::KeyStoreManager Class Reference	187
10.44QCA::Logger Class Reference	190
10.45QCA::MemoryRegion Class Reference	193
10.46QCA::MessageAuthenticationCode Class Reference	198
10.47QCA::OpenPGP Class Reference	201
10.48QCA::PasswordAsker Class Reference	203
10.49QCA::PBKDF1 Class Reference	206
10.50QCA::PBKDF2 Class Reference	208
10.51QCA::PGPKey Class Reference	210
10.52QCA::PKey Class Reference	215
10.53QCA::PrivateKey Class Reference	222
10.54QCA::PublicKey Class Reference	230
10.55QCA::QPipe Class Reference	237
10.56QCA::Random Class Reference	239
10.57QCA::RSAPrivateKey Class Reference	242
10.58QCA::RSAPublicKey Class Reference	244
10.59QCA::SASL Class Reference	246
10.60QCA::SASL::Params Class Reference	256
10.61QCA::SecureArray Class Reference	258
10.62QCA::SecureLayer Class Reference	265
10.63QCA::SecureMessage Class Reference	269
10.64QCA::SecureMessageKey Class Reference	280
10.65QCA::SecureMessageSignature Class Reference	284
10.66QCA::SecureMessageSystem Class Reference	286
10.67QCA::SymmetricKey Class Reference	288
10.68QCA::TextFilter Class Reference	290
10.69QCA::TLS Class Reference	293
10.70QCA::TLSSession Class Reference	306
10.71QCA::TokenAsker Class Reference	307
11 Qt Cryptographic Architecture File Documentation	309

11.1 qca.h File Reference	309
11.2 qca_basic.h File Reference	311
11.3 qca_cert.h File Reference	313
11.4 qca_core.h File Reference	316
11.5 qca_export.h File Reference	321
11.6 qca_keystore.h File Reference	322
11.7 qca_publickey.h File Reference	323
11.8 qca_securelayer.h File Reference	326
11.9 qca_securemessage.h File Reference	328
11.10 qca_support.h File Reference	330
11.11 qca_textfilter.h File Reference	333
11.12 qca_tools.h File Reference	335
11.13 qpipe.h File Reference	337
12 Qt Cryptographic Architecture Example Documentation	339
12.1 aes-cmac.cpp	339
12.2 base64test.cpp	345
12.3 certtest.cpp	347
12.4 ciphertest.cpp	350
12.5 cmsexample.cpp	352
12.6 eventhandlerdemo.cpp	356
12.7 hashtest.cpp	359
12.8 hextest.cpp	361
12.9 keyloader.cpp	363
12.10 mactest.cpp	365
12.11 main.cpp	367
12.12 md5crypt.cpp	368
12.13 providertest.cpp	372
12.14 publickeyexample.cpp	374
12.15 randomtest.cpp	376
12.16 rsatest.cpp	378
12.17 saslservertest.cpp	381
12.18 sasltest.cpp	387
12.19 sslservertest.cpp	393
12.20 ssltest.cpp	398
12.21 tlsocket.cpp	404

13 Qt Cryptographic Architecture Page Documentation	409
13.1 Architecture	409
13.2 Providers	412
13.3 Hashing Algorithms	413

Chapter 1

Qt Cryptographic Architecture

Taking a hint from the similarly-named [Java Cryptography Architecture](#), QCA aims to provide a straightforward and cross-platform cryptographic API, using Qt datatypes and conventions. QCA separates the API from the implementation, using plugins known as Providers. The advantage of this model is to allow applications to avoid linking to or explicitly depending on any particular cryptographic library. This allows one to easily change or upgrade Provider implementations without even needing to recompile the application!

QCA should work everywhere Qt does, including Windows/Unix/MacOSX. This version of QCA is for Qt4, and requires no Qt3 compatibility code.

1.1 Features

This library provides an easy API for the following features:

- Secure byte arrays ([QCA::SecureArray](#))
- Arbitrary precision integers ([QCA::BigInteger](#))
- Random number generation ([QCA::Random](#))
- SSL/TLS ([QCA::TLS](#))
- X509 certificates ([QCA::Certificate](#) and [QCA::CertificateCollection](#))
- X509 certificate revocation lists ([QCA::CRL](#))
- Built-in support for operating system certificate root storage ([QCA::systemStore](#))
- Simple Authentication and Security Layer (SASL) ([QCA::SASL](#))
- Cryptographic Message Syntax (e.g., for S/MIME) ([QCA::CMS](#))
- PGP messages ([QCA::OpenPGP](#))
- Unified PGP/CMS API ([QCA::SecureMessage](#))
- Subsystem for managing Smart Cards and PGP keyrings ([QCA::KeyStore](#))
- Simple but flexible logging system ([QCA::Logger](#))
- RSA ([QCA::RSAPrivateKey](#) and [QCA::RSAPublicKey](#))

- DSA ([QCA::DSAPrivateKey](#) and [QCA::DSAPublicKey](#))
- Diffie-Hellman ([QCA::DHPrivateKey](#) and [QCA::DHPublicKey](#))
- Hashing ([QCA::Hash](#)) with
 - SHA-0
 - SHA-1
 - MD2
 - MD4
 - MD5
 - RIPEMD160
 - SHA-224
 - SHA-256
 - SHA-384
 - SHA-512
- Ciphers ([QCA::Cipher](#)) using
 - BlowFish
 - Triple DES
 - DES
 - AES (128, 192 and 256 bit)
- Message Authentication Code ([QCA::MessageAuthenticationCode](#)), using
 - HMAC with SHA-1
 - HMAC with MD5
 - HMAC with RIPEMD160
 - HMAC with SHA-224
 - HMAC with SHA-256
 - HMAC with SHA-384
 - HMAC with SHA-512
- Encoding and decoding of hexadecimal ([QCA::Hex](#)) and Base64 ([QCA::Base64](#))

Functionality is supplied via plugins. This is useful for avoiding dependence on a particular crypto library and makes upgrading easier, as there is no need to recompile your application when adding or upgrading a crypto plugin. Also, by pushing crypto functionality into plugins, your application is free of legal issues, such as export regulation.

And of course, you get a very simple crypto API for Qt, where you can do things like:

```
QString hash = QCA::Hash("sha1").hashToString(blockOfData);
```

1.2 Using QCA

The application simply includes `<QtCrypto>` and links to `libqca`, which provides the 'wrapper API' and plugin loader. Crypto functionality is determined during runtime, and plugins are loaded from the 'crypto' subfolder of the Qt library paths. There are [additional examples available](#).

1.3 Availability

1.3.1 Current development

The latest version of the code is available from the KDE Subversion server (there is no formal release of the current version at this time). See <http://developer.kde.org/source/anonsvn.html> for general instructions. You do *not* need kdelibs or arts modules for QCA - just pull down kdesupport/qca. The plugins are in the same tree. Naturally you will need Qt properly set up and configured in order to build and use QCA.

The Subversion code can also be browsed [via the web](#)

1.3.2 Previous versions

A previous version of QCA (sometimes referred to as QCA1) which works with Qt3, is still available. You will need to get the main library ([qca-1.0.tar.bz2](#)) and one or more providers ([qca-tls-1.0.tar.bz2](#) for the OpenSSL based provider, or [qca-sasl-1.0.tar.bz2](#) for the SASL based provider). Note that development of QCA1 has basically stopped.

Chapter 2

Qt Cryptographic Architecture
Directory Hierarchy

2.1 Qt Cryptographic Architecture Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

examples	19
include	20
QtCrypto	21

Chapter 3

Qt Cryptographic Architecture Namespace Index

3.1 Qt Cryptographic Architecture Namespace List

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

[QCA](#) ([QCA](#) - the Qt Cryptographic Architecture) [23](#)

Chapter 4

Qt Cryptographic Architecture Hierarchical Index

4.1 Qt Cryptographic Architecture Class Hierarchy

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

QCA::Algorithm	49
QCA::Certificate	65
QCA::CertificateAuthority	74
QCA::CertificateRequest	96
QCA::Cipher	103
QCA::CRL	114
QCA::Hash	145
QCA::KeyDerivationFunction	161
QCA::PBKDF1	206
QCA::PBKDF2	208
QCA::KeyStore	172
QCA::KeyStoreEntry	177
QCA::MessageAuthenticationCode	198
QCA::PGPKey	210
QCA::PKey	215
QCA::PrivateKey	222
QCA::DHPrivateKey	122
QCA::DSAPrivateKey	129
QCA::RSAPrivateKey	242
QCA::PublicKey	230
QCA::DHPublicKey	124
QCA::DSAPublicKey	131
QCA::RSAPublicKey	244
QCA::Random	239
QCA::SASL	246
QCA::SecureMessage	269
QCA::SecureMessageSystem	286
QCA::CMS	108
QCA::OpenPGP	201
QCA::TLS	293

QCA::TLSSession	306
QCA::BigInteger	57
QCA::BufferedComputation	63
QCA::Hash	145
QCA::MessageAuthenticationCode	198
QCA::CertificateCollection	80
QCA::CertificateInfoPair	85
QCA::CertificateInfoType	87
QCA::CertificateOptions	90
QCA::ConstraintType	111
Context public QObject	
QCA::Provider	??
QCA::CRLEntry	119
QCA::DLGroup	126
QCA::Event	133
QCA::Filter	143
QCA::Cipher	103
QCA::TextFilter	290
QCA::Base64	52
QCA::Hex	150
QCA::Initializer	154
QCA::KeyBundle	155
QCA::KeyLength	167
QCA::KeyStoreInfo	185
QCA::MemoryRegion	193
QCA::SecureArray	258
QCA::InitializationVector	152
QCA::SymmetricKey	288
QCA::Provider::Context	??
QCA::BasicContext	55
QList< Certificate > [external]	
QCA::CertificateChain	77
QList< CertificateInfoPair > [external]	
QCA::CertificateInfoOrdered	84
QObject [external]	
QCA::AbstractLogDevice	47
QCA::EventHandler	138
QCA::FileWatch	141
QCA::KeyGenerator	163
QCA::KeyLoader	169
QCA::KeyStore	172
QCA::KeyStoreEntryWatcher	183
QCA::KeyStoreManager	187
QCA::Logger	190
QCA::PasswordAsker	203
QCA::SecureLayer	265
QCA::SASL	246
QCA::TLS	293
QCA::SecureMessage	269
QCA::SecureMessageSystem	286
QCA::TokenAsker	307
QCA::QPipe	237

QCA::SASL::Params	256
QCA::SecureMessageKey	280
QCA::SecureMessageSignature	284

Chapter 5

Qt Cryptographic Architecture Class Index

5.1 Qt Cryptographic Architecture Class List

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

QCA::AbstractLogDevice (An abstract log device)	47
QCA::Algorithm (General superclass for an algorithm)	49
QCA::Base64 (Base64 encoding / decoding)	52
QCA::BasicContext (Base class to use for primitive provider contexts)	55
QCA::BigInteger (Arbitrary precision integer)	57
QCA::BufferedComputation (General superclass for buffered computation algorithms)	63
QCA::Certificate (Public Key (X.509) certificate)	65
QCA::CertificateAuthority (A Certificate Authority is used to generate Certificates and Certificate Revocation Lists (CRLs))	74
QCA::CertificateChain (A chain of related Certificates)	77
QCA::CertificateCollection (Bundle of Certificates and CRLs)	80
QCA::CertificateInfoOrdered (Ordered certificate properties type)	84
QCA::CertificateInfoPair (One entry in a certificate information list)	85
QCA::CertificateInfoType (Certificate information type)	87
QCA::CertificateOptions (Certificate options)	90
QCA::CertificateRequest (Certificate Request)	96
QCA::Cipher (General class for cipher (encryption / decryption) algorithms)	103
QCA::CMS (Cryptographic Message Syntax messaging system)	108
QCA::ConstraintType (Certificate constraint)	111
QCA::CRL (Certificate Revocation List)	114
QCA::CRLEntry (Part of a CRL representing a single certificate)	119
QCA::DHPrivateKey (Diffie-Hellman Private Key)	122
QCA::DHPublicKey (Diffie-Hellman Public Key)	124
QCA::DLGroup (A discrete logarithm group)	126
QCA::DSAPrivateKey (Digital Signature Algorithm Private Key)	129
QCA::DSAPublicKey (Digital Signature Algorithm Public Key)	131
QCA::Event (An asynchronous event)	133
QCA::EventHandler (Interface class for password / passphrase / PIN and token handlers)	138
QCA::FileWatch (Support class to monitor a file for activity)	141
QCA::Filter (General superclass for filtering transformation algorithms)	143
QCA::Hash (General class for hashing algorithms)	145

QCA::Hex (Hexadecimal encoding / decoding)	150
QCA::InitializationVector (Container for initialisation vectors and nonces)	152
QCA::Initializer (Convenience method for initialising and cleaning up QCA)	154
QCA::KeyBundle (Certificate chain and private key pair)	155
QCA::KeyDerivationFunction (General superclass for key derivation algorithms)	161
QCA::KeyGenerator (Class for generating asymmetric key pairs)	163
QCA::KeyLength (Simple container for acceptable key lengths)	167
QCA::KeyLoader (Asynchronous private key loader)	169
QCA::KeyStore (General purpose key storage object)	172
QCA::KeyStoreEntry (Single entry in a KeyStore)	177
QCA::KeyStoreEntryWatcher (Class to monitor the availability of a KeyStoreEntry)	183
QCA::KeyStoreInfo (Key store information, outside of a KeyStore object)	185
QCA::KeyStoreManager (Access keystores, and monitor keystores for changes)	187
QCA::Logger (A simple logging system)	190
QCA::MemoryRegion (Array of bytes that may be optionally secured)	193
QCA::MessageAuthenticationCode (General class for message authentication code (MAC) algorithms)	198
QCA::OpenPGP (Pretty Good Privacy messaging system)	201
QCA::PasswordAsker (User password / passphrase / PIN handler)	203
QCA::PBKDF1 (Password based key derivation function version 1)	206
QCA::PBKDF2 (Password based key derivation function version 2)	208
QCA::PGPKey (Pretty Good Privacy key)	210
QCA::PKey (General superclass for public (PublicKey) and private (PrivateKey) keys used with asymmetric encryption techniques)	215
QCA::PrivateKey (Generic private key)	222
QCA::Provider (Algorithm provider)	??
QCA::Provider::Context (Internal context class used for the plugin)	??
QCA::PublicKey (Generic public key)	230
QCA::QPipe (A FIFO buffer (named pipe) abstraction)	237
QCA::Random (Source of random numbers)	239
QCA::RSAPrivateKey (RSA Private Key)	242
QCA::RSAPublicKey (RSA Public Key)	244
QCA::SASL (Simple Authentication and Security Layer protocol implementation)	246
QCA::SASL::Params (Parameter flags for the SASL authentication)	256
QCA::SecureArray (Secure array of bytes)	258
QCA::SecureLayer (Abstract interface to a security layer)	265
QCA::SecureMessage (Class representing a secure message)	269
QCA::SecureMessageKey (Key for SecureMessage system)	280
QCA::SecureMessageSignature (SecureMessage signature)	284
QCA::SecureMessageSystem (Abstract superclass for secure messaging systems)	286
QCA::SymmetricKey (Container for keys for symmetric encryption algorithms)	288
QCA::TextFilter (Superclass for text based filtering algorithms)	290
QCA::TLS (Transport Layer Security / Secure Socket Layer)	293
QCA::TLSSession (Session token, used for TLS resuming)	306
QCA::TokenAsker (User token handler)	307

Chapter 6

Qt Cryptographic Architecture File Index

6.1 Qt Cryptographic Architecture File List

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

qca.h (Summary header file for QCA)	309
qca_basic.h (Header file for classes for cryptographic primitives (basic operations))	311
qca_cert.h (Header file for PGP key and X.509 certificate related classes)	313
qca_core.h (Header file for core QCA infrastructure)	316
qca_export.h (Preprocessor magic to allow export of library symbols)	321
qca_keystore.h (Header file for classes that provide and manage keys)	322
qca_publickey.h (Header file for PublicKey and PrivateKey related classes)	323
qca_securelayer.h (Header file for SecureLayer and its subclasses)	326
qca_securemessage.h (Header file for secure message (PGP, CMS) classes)	328
qca_support.h (Header file for "support" classes used in QCA)	330
qca_textfilter.h (Header file for text encoding/decoding classes)	333
qca_tools.h (Header file for "tool" classes used in QCA)	335
qpipe.h (Header file for the QPipe FIFO class)	337

Chapter 7

Qt Cryptographic Architecture Page Index

7.1 Qt Cryptographic Architecture Related Pages

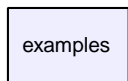
Here is a list of all related documentation pages:

Architecture	409
Providers	412
Hashing Algorithms	413

Chapter 8

Qt Cryptographic Architecture Directory Documentation

8.1 examples/ Directory Reference



Files

- file **examples.doco**

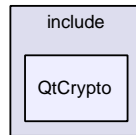
8.2 include/ Directory Reference



Directories

- directory [QtCrypto](#)

8.3 include/QtCrypto/ Directory Reference



Files

- file [qca.h](#)
- file [qca_basic.h](#)
- file [qca_cert.h](#)
- file [qca_core.h](#)
- file [qca_export.h](#)
- file [qca_keystore.h](#)
- file [qca_publickey.h](#)
- file [qca_securelayer.h](#)
- file [qca_securemessage.h](#)
- file [qca_support.h](#)
- file [qca_textfilter.h](#)
- file [qca_tools.h](#)
- file [qpipe.h](#)

Chapter 9

Qt Cryptographic Architecture Namespace Documentation

9.1 QCA Namespace Reference

[QCA](#) - the Qt Cryptographic Architecture.

Classes

- class [Random](#)
Source of random numbers.
- class [Hash](#)
General class for hashing algorithms.
- class [Cipher](#)
General class for cipher (encryption / decryption) algorithms.
- class [MessageAuthenticationCode](#)
General class for message authentication code (MAC) algorithms.
- class [KeyDerivationFunction](#)
General superclass for key derivation algorithms.
- class [PBKDF1](#)
Password based key derivation function version 1.
- class [PBKDF2](#)
Password based key derivation function version 2.
- class [CertificateInfoType](#)
Certificate information type.
- class [CertificateInfoPair](#)

One entry in a certificate information list.

- class [ConstraintType](#)
Certificate constraint.
- class [CertificateInfoOrdered](#)
Ordered certificate properties type.
- class [CertificateOptions](#)
Certificate options
- class [Certificate](#)
Public Key (X.509) certificate.
- class [CertificateChain](#)
A chain of related Certificates.
- class [CertificateRequest](#)
Certificate Request
- class [CRLEntry](#)
Part of a [CRL](#) representing a single certificate.
- class [CRL](#)
Certificate Revocation List
- class [CertificateCollection](#)
Bundle of Certificates and CRLs.
- class [CertificateAuthority](#)
A Certificate Authority is used to generate Certificates and Certificate Revocation Lists (CRLs).
- class [KeyBundle](#)
Certificate chain and private key pair.
- class [PGPKey](#)
Pretty Good Privacy key.
- class [KeyLoader](#)
Asynchronous private key loader.
- class [Initializer](#)
Convenience method for initialising and cleaning up QCA.
- class [KeyLength](#)
Simple container for acceptable key lengths.
- class **Provider**
- class [BasicContext](#)
Base class to use for primitive provider contexts.

- class [BufferedComputation](#)
General superclass for buffered computation algorithms.
- class [Filter](#)
General superclass for filtering transformation algorithms.
- class [Algorithm](#)
General superclass for an algorithm.
- class [SymmetricKey](#)
Container for keys for symmetric encryption algorithms.
- class [InitializationVector](#)
Container for initialisation vectors and nonces.
- class [Event](#)
An asynchronous event.
- class [EventHandler](#)
Interface class for password / passphrase / PIN and token handlers.
- class [PasswordAsker](#)
User password / passphrase / PIN handler.
- class [TokenAsker](#)
User token handler.
- class [KeyStoreEntry](#)
Single entry in a [KeyStore](#).
- class [KeyStoreEntryWatcher](#)
Class to monitor the availability of a [KeyStoreEntry](#).
- class [KeyStore](#)
General purpose key storage object.
- class [KeyStoreInfo](#)
Key store information, outside of a [KeyStore](#) object.
- class [KeyStoreManager](#)
Access keystores, and monitor keystores for changes.
- class [DLGroup](#)
A discrete logarithm group.
- class [PKey](#)
General superclass for public ([PublicKey](#)) and private ([PrivateKey](#)) keys used with asymmetric encryption techniques.

- class [PublicKey](#)
Generic public key.
- class [PrivateKey](#)
Generic private key.
- class [KeyGenerator](#)
Class for generating asymmetric key pairs.
- class [RSAPublicKey](#)
RSA Public Key.
- class [RSAPrivateKey](#)
RSA Private Key.
- class [DSAPublicKey](#)
Digital Signature Algorithm Public Key.
- class [DSAPrivateKey](#)
Digital Signature Algorithm Private Key.
- class [DHPublicKey](#)
Diffie-Hellman Public Key.
- class [DHPrivateKey](#)
Diffie-Hellman Private Key.
- class [SecureLayer](#)
Abstract interface to a security layer.
- class [TLSSession](#)
Session token, used for [TLS](#) resuming.
- class [TLS](#)
Transport Layer Security / Secure Socket Layer.
- class [SASL](#)
Simple Authentication and Security Layer protocol implementation.
- class [SecureMessageKey](#)
Key for [SecureMessage](#) system.
- class [SecureMessageSignature](#)
[SecureMessage](#) signature.
- class [SecureMessage](#)
Class representing a secure message.
- class [SecureMessageSystem](#)
Abstract superclass for secure messaging systems.

- class [OpenPGP](#)
Pretty Good Privacy messaging system.
- class [CMS](#)
Cryptographic Message Syntax messaging system.
- class **SyncThread**
- class **Synchronizer**
- class **DirWatch**
- class [FileWatch](#)
Support class to monitor a file for activity.
- class **Console**
- class **ConsoleReference**
- class **ConsolePrompt**
- class [Logger](#)
A simple logging system.
- class [AbstractLogDevice](#)
An abstract log device.
- class [TextFilter](#)
Superclass for text based filtering algorithms.
- class [Hex](#)
Hexadecimal encoding / decoding.
- class [Base64](#)
Base64 encoding / decoding
- class [MemoryRegion](#)
Array of bytes that may be optionally secured.
- class [SecureArray](#)
Secure array of bytes.
- class [BigInteger](#)
Arbitrary precision integer.
- class **QPipeDevice**
- class **QPipeEnd**
- class [QPipe](#)
A FIFO buffer (named pipe) abstraction.

Typedefs

- typedef **QMultiMap**< [CertificateInfoType](#), [QString](#) > [CertificateInfo](#)
- typedef **QList**< [ConstraintType](#) > [Constraints](#)
- typedef **QList**< [Provider *](#) > [ProviderList](#)
- typedef **QList**< [SecureMessageKey](#) > [SecureMessageKeyList](#)
- typedef **QList**< [SecureMessageSignature](#) > [SecureMessageSignatureList](#)

Enumerations

- enum [CertificateRequestFormat](#) { [PKCS10](#), [SPKAC](#) }
- enum [CertificateInfoTypeKnown](#) {
[CommonName](#), [Email](#), [EmailLegacy](#), [Organization](#),
[OrganizationalUnit](#), [Locality](#), [IncorporationLocality](#), [State](#),
[IncorporationState](#), [Country](#), [IncorporationCountry](#), [URI](#),
[DNS](#), [IPAddress](#), [XMPP](#) }
- enum [ConstraintTypeKnown](#) {
[DigitalSignature](#), [NonRepudiation](#), [KeyEncipherment](#), [DataEncipherment](#),
[KeyAgreement](#), [KeyCertificateSign](#), [CRLSign](#), [EncipherOnly](#),
[DecipherOnly](#), [ServerAuth](#), [ClientAuth](#), [CodeSigning](#),
[EmailProtection](#), [IPSecEndSystem](#), [IPSecTunnel](#), [IPSecUser](#),
[TimeStamping](#), [OCSPSigning](#) }
- enum [UsageMode](#) {
[UsageAny](#) = 0x00, [UsageTLSServer](#) = 0x01, [UsageTLSClient](#) = 0x02, [UsageCodeSigning](#) = 0x04,
[UsageEmailProtection](#) = 0x08, [UsageTimeStamping](#) = 0x10, [UsageCRLSigning](#) = 0x20 }
- enum [Validity](#) {
[ValidityGood](#), [ErrorRejected](#), [ErrorUntrusted](#), [ErrorSignatureFailed](#),
[ErrorInvalidCA](#), [ErrorInvalidPurpose](#), [ErrorSelfSigned](#), [ErrorRevoked](#),
[ErrorPathLengthExceeded](#), [ErrorExpired](#), [ErrorExpiredCA](#), [ErrorValidityUnknown](#) = 64 }
- enum [ValidateFlags](#) { [ValidateAll](#) = 0x00, [ValidateRevoked](#) = 0x01, [ValidateExpired](#) = 0x02,
[ValidatePolicy](#) = 0x04 }
- enum [MemoryMode](#) { [Practical](#), [Locking](#), [LockingKeepPrivileges](#) }
- enum [Direction](#) { [Encode](#), [Decode](#) }
- enum [EncryptionAlgorithm](#) { [EME_PKCS1v15](#), [EME_PKCS1_OAEP](#) }
- enum [SignatureAlgorithm](#) {
[SignatureUnknown](#), [EMSA1_SHA1](#), [EMSA3_SHA1](#), [EMSA3_MD5](#),
[EMSA3_MD2](#), [EMSA3_RIPEMD160](#), [EMSA3_Raw](#) }
- enum [SignatureFormat](#) { [DefaultFormat](#), [IEEE_1363](#), [DERSequence](#) }
- enum [PBEAlgorithm](#) {
[PBEDefault](#), [PBES2_DES_SHA1](#), [PBES2_TripleDES_SHA1](#), [PBES2_AES128_SHA1](#),
[PBES2_AES192_SHA1](#), [PBES2_AES256_SHA1](#) }
- enum [ConvertResult](#) { [ConvertGood](#), [ErrorDecode](#), [ErrorPassphrase](#), [ErrorFile](#) }
- enum [DLGroupSet](#) {
[DSA_512](#), [DSA_768](#), [DSA_1024](#), [IETF_768](#),
[IETF_1024](#), [IETF_1536](#), [IETF_2048](#), [IETF_3072](#),
[IETF_4096](#), [IETF_6144](#), [IETF_8192](#) }

- enum [SecurityLevel](#) {
[SL_None](#), [SL_Integrity](#), [SL_Export](#), [SL_Baseline](#),
[SL_High](#), [SL_Highest](#) }

Functions

- QCA_EXPORT [QString](#) [orderedToDNString](#) (const [CertificateInfoOrdered](#) &in)
- QCA_EXPORT [CertificateInfoOrdered](#) [orderedDNOnly](#) (const [CertificateInfoOrdered](#) &in)
- QCA_EXPORT [QStringList](#) [makeFriendlyNames](#) (const [QList](#)< [Certificate](#) > &list)
- QCA_EXPORT void [init](#) ()
- QCA_EXPORT void [init](#) ([MemoryMode](#) m, int prealloc)
- QCA_EXPORT void [deinit](#) ()
- QCA_EXPORT bool [haveSecureMemory](#) ()
- QCA_EXPORT bool [haveSecureRandom](#) ()
- QCA_EXPORT bool [isSupported](#) (const char *features, const [QString](#) &provider=[QString](#)())
- QCA_EXPORT bool [isSupported](#) (const [QStringList](#) &features, const [QString](#) &provider=[QString](#)())
- QCA_EXPORT [QStringList](#) [supportedFeatures](#) ()
- QCA_EXPORT [QStringList](#) [defaultFeatures](#) ()
- QCA_EXPORT bool [insertProvider](#) ([Provider](#) *p, int priority=0)
- QCA_EXPORT void [setProviderPriority](#) (const [QString](#) &name, int priority)
- QCA_EXPORT int [providerPriority](#) (const [QString](#) &name)
- QCA_EXPORT [ProviderList](#) [providers](#) ()
- QCA_EXPORT [Provider](#) * [findProvider](#) (const [QString](#) &name)
- QCA_EXPORT [Provider](#) * [defaultProvider](#) ()
- QCA_EXPORT void [scanForPlugins](#) ()
- QCA_EXPORT void [unloadAllPlugins](#) ()
- QCA_EXPORT [QString](#) [pluginDiagnosticText](#) ()
- QCA_EXPORT void [clearPluginDiagnosticText](#) ()
- QCA_EXPORT void [appendPluginDiagnosticText](#) (const [QString](#) &text)
- QCA_EXPORT void [setProperty](#) (const [QString](#) &name, const [QVariant](#) &value)
- QCA_EXPORT [QVariant](#) [getProperty](#) (const [QString](#) &name)
- QCA_EXPORT void [setProviderConfig](#) (const [QString](#) &name, const [QVariantMap](#) &config)
- QCA_EXPORT [QVariantMap](#) [getProviderConfig](#) (const [QString](#) &name)
- QCA_EXPORT void [saveProviderConfig](#) (const [QString](#) &name)
- QCA_EXPORT [QString](#) [globalRandomProvider](#) ()
- QCA_EXPORT void [setGlobalRandomProvider](#) (const [QString](#) &provider)
- QCA_EXPORT [Logger](#) * [logger](#) ()
- QCA_EXPORT bool [haveSystemStore](#) ()
- QCA_EXPORT [CertificateCollection](#) [systemStore](#) ()
- QCA_EXPORT [QString](#) [appName](#) ()
- QCA_EXPORT void [setAppName](#) (const [QString](#) &name)
- QCA_EXPORT [QString](#) [arrayToHex](#) (const [QByteArray](#) &array)
- QCA_EXPORT [QByteArray](#) [hexToArray](#) (const [QString](#) &hexString)
- QCA_EXPORT [QByteArray](#) [emsa3Encode](#) (const [QString](#) &hashName, const [QByteArray](#) &digest, int size=-1)
- QCA_EXPORT [QByteArray](#) [methodReturnType](#) (const [QMetaObject](#) *obj, const [QByteArray](#) &method, const [QList](#)< [QByteArray](#) > argTypes)
- QCA_EXPORT bool [invokeMethodWithVariants](#) ([QObject](#) *obj, const [QByteArray](#) &method, const [QVariantList](#) &args, [QVariant](#) *ret, Qt::ConnectionType type=Qt::AutoConnection)
- QCA_EXPORT const [SecureArray](#) [operator+](#) (const [SecureArray](#) &a, const [SecureArray](#) &b)

9.1.1 Detailed Description

[QCA](#) - the Qt Cryptographic Architecture.

9.1.2 Typedef Documentation

9.1.2.1 typedef QMap<[CertificateInfoType](#), QString> QCA::CertificateInfo

[Certificate](#) properties type.

With this container, the information is not necessarily stored in the same sequence as the certificate format itself. Use this container if the order the information is/was stored does not matter for you (this is the case with most applications).

Additionally, the EmailLegacy type should not be used with this container. Use Email instead.

9.1.2.2 typedef QList<[ConstraintType](#)> QCA::Constraints

Certificate constraints type

9.1.2.3 typedef QList<Provider*> QCA::ProviderList

Convenience representation for the plugin providers.

You can get a list of providers using the [providers\(\)](#) function

See also:

ProviderListIterator
[providers\(\)](#)

9.1.2.4 typedef QList<[SecureMessageKey](#)> QCA::SecureMessageKeyList

A list of message keys.

9.1.2.5 typedef QList<[SecureMessageSignature](#)> QCA::SecureMessageSignatureList

A list of signatures.

9.1.3 Enumeration Type Documentation

9.1.3.1 enum QCA::CertificateRequestFormat

[Certificate](#) Request Format.

Enumerator:

PKCS10 standard PKCS#10 format

SPKAC Signed Public Key and Challenge (Netscape) format.

9.1.3.2 enum **QCA::CertificateInfoTypeKnown**

Known types of information stored in certificates.

This enumerator offers a convenient way to work with common types.

Enumerator:

CommonName The common name (eg person), id = "2.5.4.3".

Email Email address, id = "GeneralName.rfc822Name".

EmailLegacy PKCS#9 Email field, id = "1.2.840.113549.1.9.1".

Organization An organisation (eg company), id = "2.5.4.10".

OrganizationalUnit An part of an organisation (eg a division or branch), id = "2.5.4.11".

Locality The locality (eg city, a shire, or part of a state), id = "2.5.4.7".

IncorporationLocality The locality of incorporation (EV certificates), id = "1.3.6.1.4.1.311.60.2.1.1".

State The state within the country, id = "2.5.4.8".

IncorporationState The state of incorporation (EV certificates), id = "1.3.6.1.4.1.311.60.2.1.2".

Country The country, id = "2.5.4.6".

IncorporationCountry The country of incorporation (EV certificates), id = "1.3.6.1.4.1.311.60.2.1.3".

URI Uniform Resource Identifier, id = "GeneralName.uniformResourceIdentifier".

DNS DNS name, id = "GeneralName.dNSName".

IPAddress IP address, id = "GeneralName.iPAddress".

XMPP XMPP address (see <http://www.ietf.org/rfc/rfc3920.txt>), id = "1.3.6.1.5.5.7.8.5".

9.1.3.3 enum **QCA::ConstraintTypeKnown**

Known types of certificate constraints.

This enumerator offers a convenient way to work with common types.

Enumerator:

DigitalSignature Certificate can be used to create digital signatures, id = "KeyUsage.digital-Signature"

NonRepudiation Certificate can be used for non-repudiation, id = "KeyUsage.nonRepudiation"

KeyEncipherment Certificate can be used for encrypting / decrypting keys, id = "KeyUsage.key-Encipherment"

DataEncipherment Certificate can be used for encrypting / decrypting data, id = "KeyUsage.data-Encipherment"

KeyAgreement Certificate can be used for key agreement, id = "KeyUsage.keyAgreement"

KeyCertificateSign Certificate can be used for key certificate signing, id = "KeyUsage.keyCertSign"

CRLSign Certificate can be used to sign Certificate Revocation Lists, id = "KeyUsage.crlSign"

EncipherOnly Certificate can only be used for encryption, id = "KeyUsage.encipherOnly"

DecipherOnly Certificate can only be used for decryption, id = "KeyUsage.decipherOnly"

ServerAuth Certificate can be used for server authentication (e.g. web server), id = "1.3.6.1.5.5.7.3.1". This is an extended usage constraint.

ClientAuth Certificate can be used for client authentication (e.g. web browser), id = "1.3.6.1.5.5.7.3.2". This is an extended usage constraint.

CodeSigning Certificate can be used to sign code, id = "1.3.6.1.5.5.7.3.3". This is an extended usage constraint.

EmailProtection Certificate can be used to sign / encrypt email, id = "1.3.6.1.5.5.7.3.4". This is an extended usage constraint.

IPSecEndSystem Certificate can be used to authenticate a endpoint in IPSEC, id = "1.3.6.1.5.5.7.3.5". This is an extended usage constraint.

IPSecTunnel Certificate can be used to authenticate a tunnel in IPSEC, id = "1.3.6.1.5.5.7.3.6". This is an extended usage constraint.

IPSecUser Certificate can be used to authenticate a user in IPSEC, id = "1.3.6.1.5.5.7.3.7". This is an extended usage constraint.

TimeStamping Certificate can be used to create a "time stamp" signature, id = "1.3.6.1.5.5.7.3.8". This is an extended usage constraint.

OCSPSigning Certificate can be used to sign an Online Certificate Status Protocol (OCSP) assertion, id = "1.3.6.1.5.5.7.3.9". This is an extended usage constraint.

9.1.3.4 enum [QCA::UsageMode](#)

Specify the intended usage of a certificate.

Enumerator:

- UsageAny** Any application, or unspecified.
- UsageTLSServer** server side of a [TLS](#) or SSL connection
- UsageTLSClient** client side of a [TLS](#) or SSL connection
- UsageCodeSigning** code signing certificate
- UsageEmailProtection** email (S/MIME) certificate
- UsageTimeStamping** time stamping certificate
- UsageCRLSigning** certificate revocation list signing certificate

9.1.3.5 enum [QCA::Validity](#)

The validity (or otherwise) of a certificate.

Enumerator:

- ValidityGood** The certificate is valid.
- ErrorRejected** The root CA rejected the certificate purpose.
- ErrorUntrusted** The certificate is not trusted.
- ErrorSignatureFailed** The signature does not match.
- ErrorInvalidCA** The [Certificate](#) Authority is invalid.
- ErrorInvalidPurpose** The purpose does not match the intended usage.
- ErrorSelfSigned** The certificate is self-signed, and is not found in the list of trusted certificates.

ErrorRevoked The certificate has been revoked.

ErrorPathLengthExceeded The path length from the root CA to this certificate is too long.

ErrorExpired The certificate has expired, or is not yet valid (e.g. current time is earlier than not-Before time).

ErrorExpiredCA The [Certificate](#) Authority has expired.

ErrorValidityUnknown Validity is unknown.

9.1.3.6 enum [QCA::ValidateFlags](#)

The conditions to validate for a certificate.

9.1.3.7 enum [QCA::MemoryMode](#)

Mode settings for memory allocation.

[QCA](#) can use secure memory, however most operating systems restrict the amount of memory that can be pinned by user applications, to prevent a denial-of-service attack.

[QCA](#) supports two approaches to getting memory - the mlock method, which generally requires root (administrator) level privileges, and the mmap method which is not as secure, but which should be able to be used by any process.

See also:

[Initializer](#)

Enumerator:

Practical mlock and drop root if available, else mmap

Locking mlock and drop root

LockingKeepPrivileges mlock, retaining root privileges

9.1.3.8 enum [QCA::Direction](#)

Direction settings for symmetric algorithms.

For some algorithms, it makes sense to have a "direction", such as [Cipher](#) algorithms which can be used to encrypt or decrypt.

Enumerator:

Encode Operate in the "forward" direction; for example, encrypting.

Decode Operate in the "reverse" direction; for example, decrypting.

9.1.3.9 enum [QCA::EncryptionAlgorithm](#)

Encryption algorithms.

Enumerator:

EME_PKCS1v15 Block type 2 (PKCS#1, Version 1.5).

EME_PKCS1_OAEP Optimal asymmetric encryption padding (PKCS#1, Version 2.0).

9.1.3.10 enum [QCA::SignatureAlgorithm](#)

Signature algorithm variants.

Enumerator:

- SignatureUnknown* Unknown signing algorithm.
- EMSA1_SHA1* SHA1, with EMSA1 (IEEE1363-2000) encoding (this is the usual DSA algorithm - FIPS186).
- EMSA3_SHA1* SHA1, with EMSA3 (ie PKCS#1 Version 1.5) encoding.
- EMSA3_MD5* MD5, with EMSA3 (ie PKCS#1 Version 1.5) encoding (this is the usual RSA algorithm).
- EMSA3_MD2* MD2, with EMSA3 (ie PKCS#1 Version 1.5) encoding.
- EMSA3_RIPEMD160* RIPEMD160, with EMSA3 (ie PKCS#1 Version 1.5) encoding.
- EMSA3_Raw* EMSA3 without computing a message digest or a DigestInfo encoding (identical to PKCS#11's CKM_RSA_PKCS mechanism).

9.1.3.11 enum [QCA::SignatureFormat](#)

Signature formats (DSA only).

Enumerator:

- DefaultFormat* For DSA, this is the same as IEEE_1363.
- IEEE_1363* 40-byte format from IEEE 1363 (Botan/.NET)
- DERSequence* Signature wrapped in DER formatting (OpenSSL/Java).

9.1.3.12 enum [QCA::PBEAlgorithm](#)

Password-based encryption.

Enumerator:

- PBEDefault* Use modern default (same as PBES2_TripleDES_SHA1).
- PBES2_DES_SHA1* PKCS#5 v2.0 DES/CBC,SHA1.
- PBES2_TripleDES_SHA1* PKCS#5 v2.0 TripleDES/CBC,SHA1.
- PBES2_AES128_SHA1* PKCS#5 v2.0 AES-128/CBC,SHA1.
- PBES2_AES192_SHA1* PKCS#5 v2.0 AES-192/CBC,SHA1.
- PBES2_AES256_SHA1* PKCS#5 v2.0 AES-256/CBC,SHA1.

9.1.3.13 enum [QCA::ConvertResult](#)

Return value from a format conversion.

Note that if you are checking for any result other than ConvertGood, then you may be introducing a provider specific dependency.

Enumerator:

- ConvertGood* Conversion succeeded, results should be valid.
- ErrorDecode* General failure in the decode stage.
- ErrorPassphrase* Failure because of incorrect passphrase.
- ErrorFile* Failure because of incorrect file.

9.1.3.14 enum [QCA::DLGroupSet](#)

Well known discrete logarithm group sets.

These sets are derived from three main sources: Java Cryptographic Extensions, [RFC2412](#) and [RFC3526](#).

Enumerator:

- DSA_512* 512 bit group, for compatibility with JCE
- DSA_768* 768 bit group, for compatibility with JCE
- DSA_1024* 1024 bit group, for compatibility with JCE
- IETF_768* Group 1 from RFC 2412, Section E.1.
- IETF_1024* Group 2 from RFC 2412, Section E.2.
- IETF_1536* 1536-bit MODP Group ("group 5") from RFC3526 Section 2.
- IETF_2048* 2048-bit MODP Group ("group 14") from RFC3526 Section 3.
- IETF_3072* 3072-bit MODP Group ("group 15") from RFC3526 Section 4.
- IETF_4096* 4096-bit MODP Group ("group 16") from RFC3526 Section 5.
- IETF_6144* 6144-bit MODP Group ("group 17") from RFC3526 Section 6.
- IETF_8192* 8192-bit MODP Group ("group 18") from RFC3526 Section 7.

9.1.3.15 enum [QCA::SecurityLevel](#)

Specify the lower-bound for acceptable TLS/SASL security layers.

For [TLS](#), the interpretation of these levels is:

- Any cipher suite that provides non-authenticated communications (usually anonymous Diffie-Hellman) is *SL_Integrity*.
- Any cipher suite that is limited to 40 bits (export-version crippled forms of RC2, RC4 or DES) is *SL_Export*. Standard DES (56 bits) and some forms of RC4 (64 bits) are also *SL_Export*.
- Any normal cipher (AES, Camellia, RC4 or similar) with 128 bits, or Elliptic Curve Ciphers with 283 bits, is *SL_Baseline*
- AES or Camellia at least 192 bits, triple-DES and similar ciphers are *SL_High*. ECC with 409 or more bits is also *SL_High*.
- Highest does not have an equivalent strength. It indicates that the provider should use the strongest ciphers available (but not less than *SL_High*).

Enumerator:

- SL_None* indicates that no security is ok
- SL_Integrity* must at least get integrity protection
- SL_Export* must be export level bits or more
- SL_Baseline* must be 128 bit or more
- SL_High* must be more than 128 bit
- SL_Highest* *SL_High* or max possible, whichever is greater.

9.1.4 Function Documentation

9.1.4.1 QCA_EXPORT QString QCA::orderedToDNString (const CertificateInfoOrdered & *in*)

Convert to RFC 1779 string format.

9.1.4.2 QCA_EXPORT [CertificateInfoOrdered](#) QCA::orderedDNOnly (const CertificateInfoOrdered & *in*)

Return a new [CertificateInfoOrdered](#) that only contains the Distinguished Name (DN) types found in the input object.

9.1.4.3 QCA_EXPORT QStringList QCA::makeFriendlyNames (const QList< [Certificate](#) > & *list*)

Create a list of unique friendly names among a list of certificates.

9.1.4.4 QCA_EXPORT void QCA::init ()

Initialise QCA.

This call is not normally required, because it is cleaner to use an [Initializer](#).

Examples:

[aes-cmac.cpp](#), [base64test.cpp](#), [certtest.cpp](#), [ciphertest.cpp](#), [cmsexample.cpp](#), [eventhandlerdemo.cpp](#), [hashtest.cpp](#), [hextest.cpp](#), [keyloader.cpp](#), [mactest.cpp](#), [main.cpp](#), [md5crypt.cpp](#), [providertest.cpp](#), [publickeyexample.cpp](#), [randomtest.cpp](#), [rsatest.cpp](#), [saslservtest.cpp](#), [saslttest.cpp](#), [sslservtest.cpp](#), and [ssltest.cpp](#).

9.1.4.5 QCA_EXPORT void QCA::init ([MemoryMode](#) *m*, int *prealloc*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

m the MemoryMode to use

prealloc the amount of memory in kilobytes to allocate for secure storage

9.1.4.6 QCA_EXPORT void QCA::deinit ()

Clean up routine.

This routine cleans up QCA, including memory allocations This call is not normally required, because it is cleaner to use an [Initializer](#)

9.1.4.7 QCA_EXPORT bool QCA::haveSecureMemory ()

Test if secure storage memory is available.

Returns:

true if secure storage memory is available

9.1.4.8 QCA_EXPORT bool QCA::haveSecureRandom ()

Test if secure random is available.

Secure random is considered available if the global random provider is not the default provider.

Returns:

true if secure random is available

9.1.4.9 QCA_EXPORT bool QCA::isSupported (const char * *features*, const QString & *provider* = QString ())

Test if a capability (algorithm) is available.

Since capabilities are made available at runtime, you should always check before using a capability the first time, as shown below.

```
QCA::init();
if(!QCA::isSupported("sha1"))
    printf("SHA1 not supported!\n");
else
{
    QString result = QCA::SHA1::hashToString(myString);
    printf("sha1(\"%s\") = [%s]\n", myString.data(), qPrintable(result));
}
```

Parameters:

features the name of the capability to test for

provider if specified, only check for the capability in that specific provider. If not provided, or provided as an empty string, then check for capabilities in all available providers

Returns:

true if the capability is available, otherwise false

Note that you can test for a combination of capabilities, using a comma delimited list:

```
QCA::isSupported("sha1,md5"):
```

which will return true if all of the capabilities listed are present.

Examples:

[aes-cmac.cpp](#), [certtest.cpp](#), [ciphertest.cpp](#), [cmsexample.cpp](#), [hashtest.cpp](#), [mactest.cpp](#), [md5crypt.cpp](#), [publickeyexample.cpp](#), [rsatest.cpp](#), [saslservtest.cpp](#), [saslttest.cpp](#), [sslservtest.cpp](#), and [ssltest.cpp](#).

9.1.4.10 QCA_EXPORT bool QCA::isSupported (const QStringList & *features*, const QString & *provider* = QString ())

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

features a list of features to test for

provider if specified, only check for the capability in that specific provider. If not provided, or provided as an empty string, then check for capabilities in all available providers

9.1.4.11 QCA_EXPORT QStringList QCA::supportedFeatures ()

Generate a list of all the supported features in plugins, and in built in capabilities.

Returns:

a list containing the names of the features

The following code writes a list of features to standard out

```
QStringList capabilities;  
capabilities = QCA::supportedFeatures();  
std::cout << "Supported:" << capabilities.join(",") << std::endl;
```

See also:

[isSupported\(const char *features\)](#)

[isSupported\(const QStringList &features\)](#)

[defaultFeatures\(\)](#)

9.1.4.12 QCA_EXPORT QStringList QCA::defaultFeatures ()

Generate a list of the built in features.

This differs from [supportedFeatures\(\)](#) in that it does not include features provided by plugins.

Returns:

a list containing the names of the features

The following code writes a list of features to standard out

```
QStringList capabilities;  
capabilities = QCA::defaultFeatures();  
std::cout << "Default:" << capabilities.join(",") << std::endl;
```

See also:

[isSupported](#)

[supportedFeatures\(\)](#)

Examples:

[providertest.cpp](#).

9.1.4.13 QCA_EXPORT bool QCA::insertProvider (Provider *p, int priority = 0)

Add a provider to the current list of providers.

This function allows you to add a provider to the current plugin providers at a specified priority. If a provider with the name already exists, this call fails.

Parameters:

- p* a pointer to a [Provider](#) object, which must be set up.
- priority* the priority level to set the provider to

Returns:

true if the provider is added, and false if the provider is not added (failure)

See also:

[setProviderPriority](#) for a description of the provider priority system

Examples:

[aes-cmac.cpp](#).

9.1.4.14 QCA_EXPORT void QCA::setProviderPriority (const QString & name, int priority)

Change the priority of a specified provider.

[QCA](#) supports a number of providers, and if a number of providers support the same algorithm, it needs to choose between them. You can do this at object instantiation time (by specifying the name of the provider that should be used). Alternatively, you can provide a relative priority level at an application level, using this call.

Priority is used at object instantiation time. The provider is selected according to the following logic:

- if a particular provider is nominated, and that provider supports the required algorithm, then the nominated provider is used
- if no provider is nominated, or it doesn't support the required algorithm, then the provider with the lowest priority number will be used, if that provider supports the algorithm.
- if the provider with the lowest priority number doesn't support the required algorithm, the provider with the next lowest priority number will be tried, and so on through to the provider with the largest priority number
- if none of the plugin providers support the required algorithm, then the default (built-in) provider will be tried.

Parameters:

- name* the name of the provider
- priority* the new priority of the provider. As a special case, if you pass in -1, then this provider gets the same priority as the the last provider that was added or had its priority set using this call.

See also:

[providerPriority](#)

9.1.4.15 QCA_EXPORT int QCA::providerPriority (const QString & name)

Return the priority of a specified provider.

The name of the provider (eg "qca-openssl") is used to look up the current priority associated with that provider. If the provider is not found (or something else went wrong), -1 is returned.

Parameters:

name the name of the provider

Returns:

the current priority level

See also:

[setProviderPriority](#) for a description of the provider priority system

9.1.4.16 QCA_EXPORT ProviderList QCA::providers ()

Return a list of the current providers.

The current plugin providers are provided as a list, which you can iterate over using ProviderListIterator.

See also:

[ProviderList](#)

ProviderListIterator

Examples:

[providertest.cpp](#).

9.1.4.17 QCA_EXPORT Provider* QCA::findProvider (const QString & name)

Return the named provider, or 0 if not found.

9.1.4.18 QCA_EXPORT Provider* QCA::defaultProvider ()

Return the default provider.

9.1.4.19 QCA_EXPORT void QCA::scanForPlugins ()

Scan for new plugins.

Examples:

[providertest.cpp](#).

9.1.4.20 QCA_EXPORT void QCA::unloadAllPlugins ()

Unload the current plugins.

9.1.4.21 QCA_EXPORT QString QCA::pluginDiagnosticText ()

Retrieve plugin diagnostic text.

9.1.4.22 QCA_EXPORT void QCA::clearPluginDiagnosticText ()

Clear plugin diagnostic text.

9.1.4.23 QCA_EXPORT void QCA::appendPluginDiagnosticText (const QString & text)

Add plugin diagnostic text.

This function should only be called by providers.

9.1.4.24 QCA_EXPORT void QCA::setProperty (const QString & name, const QVariant & value)

Set a global property.

9.1.4.25 QCA_EXPORT QVariant QCA::getProperty (const QString & name)

Retrieve a global property.

9.1.4.26 QCA_EXPORT void QCA::setProviderConfig (const QString & name, const QVariantMap & config)

Set provider configuration.

Allowed value types: **QString**, int, bool

9.1.4.27 QCA_EXPORT QVariantMap QCA::getProviderConfig (const QString & name)

Retrieve provider configuration.

9.1.4.28 QCA_EXPORT void QCA::saveProviderConfig (const QString & name)

Save provider configuration to persistent storage.

9.1.4.29 QCA_EXPORT QString QCA::globalRandomProvider ()

Return the name of the global random number provider.

9.1.4.30 QCA_EXPORT void QCA::setGlobalRandomProvider (const QString & provider)

Change the global random number provider.

The [Random](#) capabilities of QCA are provided as part of the built in capabilities, however the generator can be changed if required.

9.1.4.31 QCA_EXPORT [Logger*](#) QCA::logger ()

Return a reference to the QCA [Logger](#), which is used for diagnostics and error recording.

The system [Logger](#) is automatically created for you on start.

9.1.4.32 QCA_EXPORT bool QCA::haveSystemStore ()

Test if [QCA](#) can access the root CA certificates.

If root certificates are available, this function returns true, otherwise it returns false.

See also:

[systemStore](#)

Examples:

[certtest.cpp](#), and [ssltest.cpp](#).

9.1.4.33 QCA_EXPORT [CertificateCollection](#) QCA::systemStore ()

Get system-wide root [Certificate](#) Authority (CA) certificates.

Many operating systems (or distributions, on Linux-type systems) come with some trusted certificates. Typically, these include the root certificates for major [Certificate](#) Authorities (for example, Verisign, Comodo) and some additional certificates that are used for system updates. They are provided in different ways for different systems.

This function provides a common way to access the system certificates. There are other ways to access certificates - see the various I/O methods (such as [fromDER\(\)](#) and [fromPEM\(\)](#)) in the [Certificate](#) and [CertificateCollection](#) classes.

Note:

Availability of the system certificates depends on how QCA was built. You can test whether the system certificates are available using the [haveSystemStore\(\)](#) function.

Examples:

[certtest.cpp](#), [ssltest.cpp](#), and [tlssocket.cpp](#).

9.1.4.34 QCA_EXPORT QString QCA::appName ()

Get the application name that will be used by [SASL](#) server mode.

The application name is used by [SASL](#) in server mode, as some systems might have different security policies depending on the app. The default application name is 'qca'

9.1.4.35 QCA_EXPORT void QCA::setAppName (const QString & name)

Set the application name that will be used by [SASL](#) server mode.

The application name is used by [SASL](#) in server mode, as some systems might have different security policies depending on the app. This should be set before using [SASL](#) objects, and it cannot be changed later.

Parameters:

name the name string to use for [SASL](#) server mode

Examples:

[saslservtest.cpp](#).

9.1.4.36 QCA_EXPORT QString QCA::arrayToHex (const QByteArray & array)

Convert a byte array to printable hexadecimal representation.

This is a convenience function to convert an arbitrary **QByteArray** to a printable representation.

```
QByteArray test(10);
test.fill('a');
// 0x61 is 'a' in ASCII
if (QString("61616161616161616161") == QCA::arrayToHex(test) )
{
    printf ("arrayToHex passed\n");
}
```

Parameters:

array the array to be converted

Returns:

a printable representation

Examples:

[aes-cmac.cpp](#), [ciphertest.cpp](#), [hashtest.cpp](#), [mactest.cpp](#), and [rsatest.cpp](#).

9.1.4.37 QCA_EXPORT QByteArray QCA::hexToArray (const QString & hexString)

Convert a **QString** containing a hexadecimal representation of a byte array into a **QByteArray**.

This is a convenience function to convert a printable representation into a **QByteArray** - effectively the inverse of [QCA::arrayToHex](#).

```
QCA::init();
QByteArray test(10);

test.fill('b'); // 0x62 in hexadecimal
test[7] = 0x00; // can handle strings with nulls

if (QCA::hexToArray(QString("6262626262626262006262")) == test )
{
    printf ("hexToArray passed\n");
}
```

Parameters:

hexString the string containing a printable representation to be converted

Returns:

the equivalent **QByteArray**

Examples:

[aes-cmac.cpp](#).

9.1.4.38 QCA_EXPORT QByteArray QCA::emsa3Encode (const QString & hashName, const QByteArray & digest, int size = -1)

Encode a hash result in EMSA3 (PKCS#1) format.

This is a convenience function for providers that only have access to raw RSA signing (mainly smartcard providers). This is a built-in function of **QCA** and does not utilize a provider. SHA1, MD5, MD2, and RIPEMD160 are supported.

9.1.4.39 QCA_EXPORT QByteArray QCA::methodReturnType (const QMetaObject * *obj*, const QByteArray & *method*, const QList< QByteArray > *argTypes*)

Convenience method to determine the return type of a method.

This function identifies the return type of a specified method. This function can be used as shown:

```
class TestClass : public QObject
{
    Q_OBJECT
    // ...
public slots:
    QString qstringMethod() { return QString(); };
    bool boolMethod( const QString & ) { return true; };
};

QByteArray myTypeName;

TestClass testClass;
QList<QByteArray> argsList; // empty list, since no args

myTypeName = QCA::methodReturnType( testClass.metaObject(), QByteArray( "qstringMethod" ), argsList );
// myTypeName is "QString"

myTypeName = QCA::methodReturnType( testClass.metaObject(), QByteArray( "boolMethod" ), argsList );
// myTypeName is "", because there is no method called "boolMethod" that has no arguments

argsList << "QString"; // now we have one argument
myTypeName = QCA::methodReturnType( testClass.metaObject(), QByteArray( "boolMethod" ), argsList );
// myTypeName is "bool"
```

The return type name of a method returning void is an empty string, not "void"

Note:

This function is not normally required for use with QCA. It is provided for use in your code, if required.

Parameters:

- obj* the **QMetaObject** for the object
- method* the name of the method (without the arguments or brackets)
- argTypes* the list of argument types of the method

Returns:

the name of the type that this method will return with the specified argument types.

See also:

QMetaType for more information on the Qt meta type system.

9.1.4.40 QCA_EXPORT bool QCA::invokeMethodWithVariants (QObject * *obj*, const QByteArray & *method*, const QVariantList & *args*, QVariant * *ret*, Qt::ConnectionType *type* = Qt::AutoConnection)

Convenience method to invoke a method by name, using a variant list of arguments.

This function can be used as shown:

```
class TestClass : public QObject
{
```

```

    Q_OBJECT
    // ...
public slots:
    QString qstringMethod() { return QString( "the result" ); };
    bool boolMethod( const QString & ) { return true; };
};

TestClass *testClass = new TestClass;
QVariantList args;

QVariant stringRes;
// calls testClass->qstringMethod() with no arguments ( since args is an empty list)
bool ret = QCA::invokeMethodWithVariants( testClass, QByteArray( "qstringMethod" ), args, &stringRes );
// ret is true (since call succeeded), stringRes.toString() is a string - "the result"

QVariant boolResult;
QString someString( "not important" );
args << someString;
// calls testClass->boolMethod( someString ), returning result in boolResult
ret = QCA::invokeMethodWithVariants( testClass, QByteArray( "boolMethod" ), args, &boolResult );
// ret is true (since call succeeded), boolResult.toBool() is true.

```

Parameters:

obj the object to call the method on
method the name of the method (without the arguments or brackets)
args the list of arguments to use in the method call
ret the return value of the method (unchanged if the call fails)
type the type of connection to use

Returns:

true if the call succeeded, otherwise false

9.1.4.41 QCA_EXPORT const [SecureArray](#) QCA::operator+ (const [SecureArray](#) & *a*, const [SecureArray](#) & *b*)

Returns an array that is the result of concatenating *a* and *b*.

Chapter 10

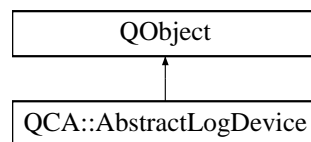
Qt Cryptographic Architecture Class Documentation

10.1 QCA::AbstractLogDevice Class Reference

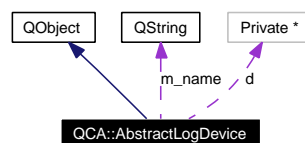
An abstract log device.

```
#include <qca_support.h>
```

Inheritance diagram for QCA::AbstractLogDevice::



Collaboration diagram for QCA::AbstractLogDevice:



Public Member Functions

- `QString name () const`
- virtual void `logTextMessage` (const `QString` &message, enum `Logger::Severity` severity)
- virtual void `logBinaryMessage` (const `QByteArray` &blob, `Logger::Severity` severity)

Protected Member Functions

- `AbstractLogDevice` (const `QString` &name, `QObject` *parent=0)

10.1.1 Detailed Description

An abstract log device.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 `QCA::AbstractLogDevice::AbstractLogDevice (const QString & name, QObject * parent = 0) [explicit, protected]`

Create a new message logger.

Parameters:

name the name of this log device

parent the parent for this logger

10.1.3 Member Function Documentation

10.1.3.1 `QString QCA::AbstractLogDevice::name () const`

The name of this log device.

10.1.3.2 `virtual void QCA::AbstractLogDevice::logTextMessage (const QString & message, enum Logger::Severity severity) [virtual]`

Log a message.

The default implementation does nothing - you should override this method in your subclass to do whatever logging is required

10.1.3.3 `virtual void QCA::AbstractLogDevice::logBinaryMessage (const QByteArray & blob, Logger::Severity severity) [virtual]`

Log a binary blob.

The default implementation does nothing - you should override this method in your subclass to do whatever logging is required

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

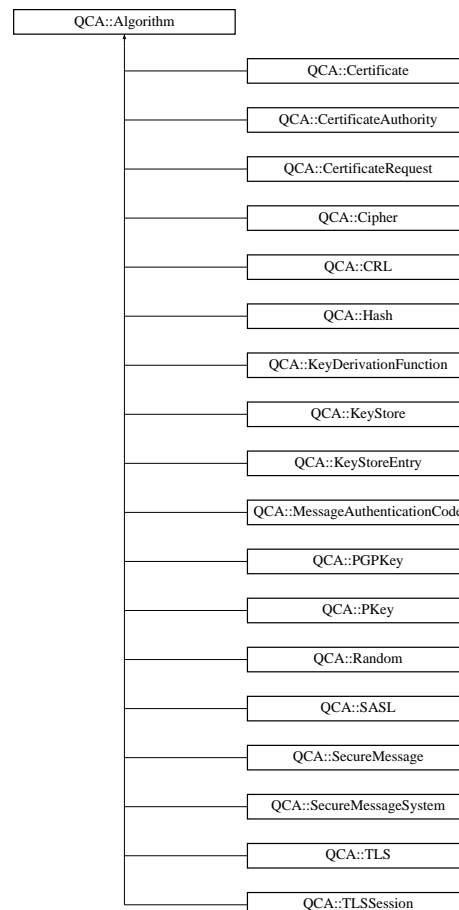
- [qca_support.h](#)

10.2 QCA::Algorithm Class Reference

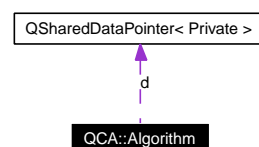
General superclass for an algorithm.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Algorithm::



Collaboration diagram for QCA::Algorithm:



Public Member Functions

- [Algorithm](#) (const [Algorithm](#) &from)
- [Algorithm](#) & [operator=](#) (const [Algorithm](#) &from)
- [QString](#) [type](#) () const
- Provider * [provider](#) () const

- [Provider::Context](#) * [context](#) ()
- const [Provider::Context](#) * [context](#) () const
- void [change](#) ([Provider::Context](#) *c)
- void [change](#) (const [QString](#) &type, const [QString](#) &provider)
- [Provider::Context](#) * [takeContext](#) ()

Protected Member Functions

- [Algorithm](#) ()
- [Algorithm](#) (const [QString](#) &type, const [QString](#) &provider)

10.2.1 Detailed Description

General superclass for an algorithm.

This is a fairly abstract class, mainly used for implementing the backend "provider" interface.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 [QCA::Algorithm::Algorithm](#) (const [Algorithm](#) &*from*)

Standard copy constructor.

Parameters:

from the [Algorithm](#) to copy from

10.2.2.2 [QCA::Algorithm::Algorithm](#) () [protected]

Constructor for empty algorithm.

10.2.2.3 [QCA::Algorithm::Algorithm](#) (const [QString](#) &*type*, const [QString](#) &*provider*) [protected]

Constructor of a particular algorithm.

Parameters:

type the algorithm to construct

provider the name of a particular [Provider](#)

10.2.3 Member Function Documentation

10.2.3.1 [Algorithm](#)& [QCA::Algorithm::operator=](#) (const [Algorithm](#) &*from*)

Assignment operator.

Parameters:

from the [Algorithm](#) to copy state from

10.2.3.2 QString QCA::Algorithm::type () const

The name of the algorithm type.

Reimplemented in [QCA::Hash](#), [QCA::Cipher](#), [QCA::MessageAuthenticationCode](#), [QCA::KeyStoreEntry](#), [QCA::KeyStore](#), [QCA::PKey](#), and [QCA::SecureMessage](#).

10.2.3.3 Provider* QCA::Algorithm::provider () const

The name of the provider.

Each algorithm is implemented by a provider. This allows you to figure out which provider is associated

10.2.3.4 Provider::Context* QCA::Algorithm::context ()

For internal use only.

The context associated with this algorithm

10.2.3.5 const Provider::Context* QCA::Algorithm::context () const

For internal use only.

The context associated with this algorithm

10.2.3.6 void QCA::Algorithm::change (Provider::Context * c)

For internal use only.

Set the [Provider](#) for this algorithm

Parameters:

c the context for the [Provider](#) to use

10.2.3.7 void QCA::Algorithm::change (const QString & type, const QString & provider)

For internal use only.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

type the name of the algorithm to use

provider the name of the preferred provider

10.2.3.8 Provider::Context* QCA::Algorithm::takeContext ()

For internal use only.

Take the [Provider](#) from this algorithm

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

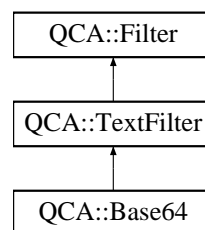
- [qca_core.h](#)

10.3 QCA::Base64 Class Reference

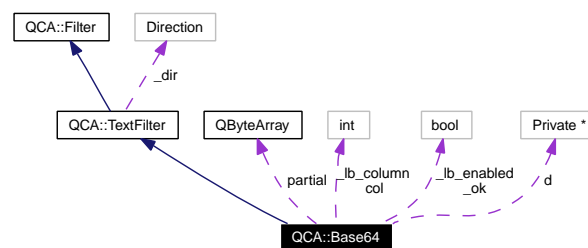
Base64 encoding / decoding

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Base64::



Collaboration diagram for QCA::Base64:



Public Member Functions

- [Base64](#) ([Direction](#) dir=Encode)
- bool [lineBreaksEnabled](#) () const
- int [lineBreaksColumn](#) () const
- void [setLineBreaksEnabled](#) (bool b)
- void [setLineBreaksColumn](#) (int column)
- virtual void [clear](#) ()
- virtual [MemoryRegion](#) [update](#) (const [MemoryRegion](#) &a)
- virtual [MemoryRegion](#) [final](#) ()
- virtual bool [ok](#) () const

10.3.1 Detailed Description

Base64 encoding / decoding

Examples:

[base64test.cpp](#), [cmsexample.cpp](#), [publickeyexample.cpp](#), [saslservtest.cpp](#), and [saslttest.cpp](#).

10.3.2 Constructor & Destructor Documentation

10.3.2.1 QCA::Base64::Base64 ([Direction](#) *dir* = Encode)

Standard constructor.

Parameters:

dir the Direction that should be used.

Note:

The direction can be changed using the [setup\(\)](#) call.

10.3.3 Member Function Documentation

10.3.3.1 bool QCA::Base64::lineBreaksEnabled () const

Returns true if line breaks are enabled.

10.3.3.2 int QCA::Base64::lineBreaksColumn () const

Returns the line break column.

10.3.3.3 void QCA::Base64::setLineBreaksEnabled (bool *b*)

Sets line break mode.

If enabled, linebreaks will be added to encoded output or accepted in encoded input. If disabled, linebreaks in encoded input will cause a failure to decode. The default is disabled.

10.3.3.4 void QCA::Base64::setLineBreaksColumn (int *column*)

Sets the column that linebreaks should be inserted at when encoding.

10.3.3.5 virtual void QCA::Base64::clear () [virtual]

Reset the internal state.

This is useful to reuse an existing [Base64](#) object

Implements [QCA::Filter](#).

10.3.3.6 virtual [MemoryRegion](#) QCA::Base64::update (const [MemoryRegion](#) & *a*) [virtual]

Process more data, returning the corresponding encoded or decoded (depending on the Direction set in the constructor or [setup\(\)](#) call) representation.

If you find yourself with code that only calls this method once, you might be better off using [encode\(\)](#) or [decode\(\)](#). Similarly, if the data is really a string, you might be better off using [arrayToString\(\)](#), [encode-String\(\)](#), [stringToArray\(\)](#) or [decodeString\(\)](#).

Parameters:

a the array containing data to process

Implements [QCA::Filter](#).

10.3.3.7 virtual [MemoryRegion](#) QCA::Base64::final () [virtual]

Complete the algorithm.

Returns:

any remaining output. Because of the way [Base64](#) encoding works, you will get either an empty array, or an array containing one or two "=" (equals, 0x3D) characters.

Implements [QCA::Filter](#).

10.3.3.8 virtual bool QCA::Base64::ok () const [virtual]

Test if an [update\(\)](#) or [final\(\)](#) call succeeded.

Returns:

true if the previous call succeeded

Implements [QCA::Filter](#).

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

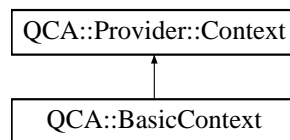
- [qca_textfilter.h](#)

10.4 QCA::BasicContext Class Reference

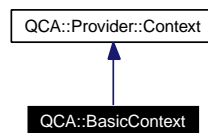
Base class to use for primitive provider contexts.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::BasicContext::



Collaboration diagram for QCA::BasicContext:



Protected Member Functions

- [BasicContext](#) (Provider *parent, const [QString](#) &type)
- [BasicContext](#) (const [BasicContext](#) &from)

10.4.1 Detailed Description

Base class to use for primitive provider contexts.

For internal use only.

This class inherits [Provider::Context](#) and calls `moveToThread(0)` on itself, thereby disabling the event properties of the underlying **QObject**. Context types that need to be a **QObject** should inherit from [Provider::Context](#), those that don't should inherit from [BasicContext](#).

10.4.2 Constructor & Destructor Documentation

10.4.2.1 QCA::BasicContext::BasicContext (Provider *parent, const [QString](#) &type) [protected]

Standard constructor.

Parameters:

parent the parent provider for this context

type the name of the provider context type

10.4.2.2 QCA::BasicContext::BasicContext (const [BasicContext](#) &*from*) [protected]

Copy constructor.

Parameters:

from the Context to copy from

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

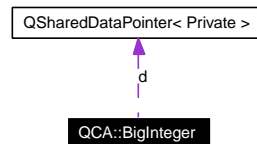
- [qca_core.h](#)

10.5 QCA::BigInteger Class Reference

Arbitrary precision integer.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::BigInteger:



Public Member Functions

- [BigInteger](#) ()
- [BigInteger](#) (int n)
- [BigInteger](#) (const char *c)
- [BigInteger](#) (const [QString](#) &s)
- [BigInteger](#) (const [QCA::SecureArray](#) &a)
- [BigInteger](#) (const [BigInteger](#) &from)
- [BigInteger](#) & [operator=](#) (const [BigInteger](#) &from)
- [BigInteger](#) & [operator=](#) (const [QString](#) &s)
- [BigInteger](#) & [operator+=](#) (const [BigInteger](#) &b)
- [BigInteger](#) & [operator-=](#) (const [BigInteger](#) &b)
- [QCA::SecureArray toArray](#) () const
- void [fromArray](#) (const [QCA::SecureArray](#) &a)
- [QString toString](#) () const
- bool [fromString](#) (const [QString](#) &s)
- int [compare](#) (const [BigInteger](#) &n) const
- bool [operator==](#) (const [BigInteger](#) &other) const
- bool [operator!=](#) (const [BigInteger](#) &other) const
- bool [operator<=](#) (const [BigInteger](#) &other) const
- bool [operator>=](#) (const [BigInteger](#) &other) const
- bool [operator<](#) (const [BigInteger](#) &other) const
- bool [operator>](#) (const [BigInteger](#) &other) const

Related Functions

(Note that these are not member functions.)

- `QCA_EXPORT QTextStream & operator<< (QTextStream &stream, const BigInteger &b)`

10.5.1 Detailed Description

Arbitrary precision integer.

[BigInteger](#) provides arbitrary precision integers.

```
if ( BigInteger("3499543804349") ==  
    BigInteger("38493290803248") + BigInteger( 343 ) )  
{  
    // do something  
}
```

10.5.2 Constructor & Destructor Documentation

10.5.2.1 QCA::BigInteger::BigInteger ()

Constructor.

Creates a new [BigInteger](#), initialised to zero.

10.5.2.2 QCA::BigInteger::BigInteger (int *n*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

n an alternative integer initialisation value.

10.5.2.3 QCA::BigInteger::BigInteger (const char * *c*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

c an alternative initialisation value, encoded as a character array

```
BigInteger b ( "9890343" );
```

10.5.2.4 QCA::BigInteger::BigInteger (const QString & *s*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

s an alternative initialisation value, encoded as a string

10.5.2.5 QCA::BigInteger::BigInteger (const [QCA::SecureArray](#) & *a*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

a an alternative initialisation value, encoded as [SecureArray](#)

10.5.2.6 QCA::BigInteger::BigInteger (const [BigInteger](#) & *from*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

from an alternative initialisation value, encoded as a BigInteger

10.5.3 Member Function Documentation

10.5.3.1 [BigInteger](#)& QCA::BigInteger::operator= (const [BigInteger](#) & *from*)

Assignment operator.

Parameters:

from the [BigInteger](#) to copy from

```
BigInteger a; // a is zero
BigInteger b( 500 );
a = b; // a is now 500
```

10.5.3.2 [BigInteger](#)& QCA::BigInteger::operator= (const QString & *s*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

s the [QString](#) containing an integer representation

See also:

bool [fromString](#)(const [QString](#) &*s*)

Note:

it is the application's responsibility to make sure that the [QString](#) represents a valid integer (ie it only contains numbers and an optional minus sign at the start)

10.5.3.3 [BigInteger](#)& QCA::BigInteger::operator+= (const [BigInteger](#) & *b*)

Increment in place operator.

Parameters:

b the amount to increment by

```
BigInteger a; // a is zero
BigInteger b( 500 );
a += b; // a is now 500
a += b; // a is now 1000
```

10.5.3.4 `BigInteger` & `QCA::BigInteger::operator-= (const BigInteger & b)`

Decrement in place operator.

Parameters:

b the amount to decrement by

```
BigInteger a; // a is zero
BigInteger b( 500 );
a -= b; // a is now -500
a -= b; // a is now -1000
```

10.5.3.5 `QCA::SecureArray` `QCA::BigInteger::toArray () const`

Output `BigInteger` as a byte array, useful for storage or transmission.

The format is a binary integer in sign-extended network-byte-order.

See also:

void `fromArray(const SecureArray &a);`

10.5.3.6 void `QCA::BigInteger::fromArray (const QCA::SecureArray & a)`

Assign from an array.

The input is expected to be a binary integer in sign-extended network-byte-order.

Parameters:

a a `SecureArray` that represents an integer

See also:

`BigInteger(const SecureArray &a);`
`SecureArray toArray() const;`

10.5.3.7 `QString` `QCA::BigInteger::toString () const`

Convert `BigInteger` to a `QString`.

```
QString aString;
BigInteger aBiggishInteger( 5878990 );
aString = aBiggishInteger.toString(); // aString is now "5878990"
```

10.5.3.8 bool `QCA::BigInteger::fromString (const QString & s)`

Assign from a `QString`.

Parameters:

s a `QString` that represents an integer

Note:

it is the application's responsibility to make sure that the **QString** represents a valid integer (ie it only contains numbers and an optional minus sign at the start)

See also:

[BigInteger\(const QString &s\)](#)
[BigInteger & operator=\(const QString &s\)](#)

10.5.3.9 int QCA::BigInteger::compare (const [BigInteger](#) & n) const

Compare this value with another [BigInteger](#).

Normally it is more readable to use one of the operator overloads, so you don't need to use this method directly.

Parameters:

n the [BigInteger](#) to compare with

Returns:

zero if the values are the same, negative if the argument is less than the value of this [BigInteger](#), and positive if the argument value is greater than this [BigInteger](#)

```
BigInteger a( "400" );
BigInteger b( "-400" );
BigInteger c( " 200 " );
int result;
result = a.compare( b );           // return positive 400 > -400
result = a.compare( c );           // return positive, 400 > 200
result = b.compare( c );           // return negative, -400 < 200
```

10.5.3.10 bool QCA::BigInteger::operator== (const [BigInteger](#) & other) const [inline]

Equality operator.

Returns true if the two [BigInteger](#) values are the same, including having the same sign.

10.5.3.11 bool QCA::BigInteger::operator!= (const [BigInteger](#) & other) const [inline]

Inequality operator.

Returns true if the two [BigInteger](#) values are different in magnitude, sign or both.

10.5.3.12 bool QCA::BigInteger::operator<= (const [BigInteger](#) & other) const [inline]

Less than or equal operator.

Returns true if the [BigInteger](#) value on the left hand side is equal to or less than the [BigInteger](#) value on the right hand side.

10.5.3.13 bool QCA::BigInteger::operator>= (const [BigInteger](#) & other) const [inline]

Greater than or equal operator.

Returns true if the [BigInteger](#) value on the left hand side is equal to or greater than the [BigInteger](#) value on the right hand side.

10.5.3.14 `bool QCA::BigInteger::operator< (const BigInteger & other) const` `[inline]`

Less than operator.

Returns true if the [BigInteger](#) value on the left hand side is less than the [BigInteger](#) value on the right hand side.

10.5.3.15 `bool QCA::BigInteger::operator> (const BigInteger & other) const` `[inline]`

Greater than operator.

Returns true if the [BigInteger](#) value on the left hand side is greater than the [BigInteger](#) value on the right hand side.

10.5.4 Friends And Related Function Documentation**10.5.4.1** `QCA_EXPORT QTextStream & operator<< (QTextStream & stream, const BigInteger & b)` `[related]`

Stream operator.

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

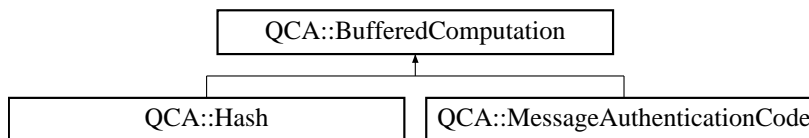
- [qca_tools.h](#)

10.6 QCA::BufferedComputation Class Reference

General superclass for buffered computation algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::BufferedComputation::



Public Member Functions

- virtual void [clear](#) ()=0
- virtual void [update](#) (const [MemoryRegion](#) &a)=0
- virtual [MemoryRegion](#) [final](#) ()=0
- [MemoryRegion](#) [process](#) (const [MemoryRegion](#) &a)

10.6.1 Detailed Description

General superclass for buffered computation algorithms.

A buffered computation is characterised by having the algorithm take data in an incremental way, then having the results delivered at the end. Conceptually, the algorithm has some internal state that is modified when you call [update\(\)](#) and returned when you call [final\(\)](#).

10.6.2 Member Function Documentation

10.6.2.1 virtual void QCA::BufferedComputation::clear () [pure virtual]

Reset the internal state.

Implemented in [QCA::Hash](#), and [QCA::MessageAuthenticationCode](#).

10.6.2.2 virtual void QCA::BufferedComputation::update (const [MemoryRegion](#) &a) [pure virtual]

Update the internal state with a byte array.

Parameters:

a the byte array of data that is to be used to update the internal state.

Implemented in [QCA::Hash](#), and [QCA::MessageAuthenticationCode](#).

10.6.2.3 virtual [MemoryRegion](#) QCA::BufferedComputation::final () [pure virtual]

Complete the algorithm and return the internal state.

Implemented in [QCA::Hash](#), and [QCA::MessageAuthenticationCode](#).

10.6.2.4 [MemoryRegion](#) QCA::BufferedComputation::process (const [MemoryRegion](#) & a)

Perform an "all in one" update, returning the result.

This is appropriate if you have all the data in one array - just call process on that array, and you will get back the results of the computation.

Note:

This will invalidate any previous computation using this object.

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

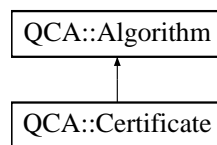
- [qca_core.h](#)

10.7 QCA::Certificate Class Reference

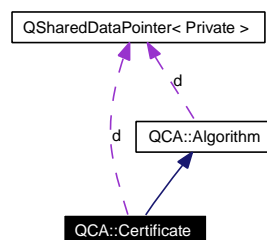
Public Key (X.509) certificate.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Certificate::



Collaboration diagram for QCA::Certificate:



Public Member Functions

- [Certificate](#) ()
- [Certificate](#) (const **QString** &fileName)
- [Certificate](#) (const [CertificateOptions](#) &opts, const [PrivateKey](#) &key, const **QString** &provider=**QString**())
- [Certificate](#) (const [Certificate](#) &from)
- [Certificate](#) & operator= (const [Certificate](#) &from)
- bool [isNull](#) () const
- **QDateTime** [notValidBefore](#) () const
- **QDateTime** [notValidAfter](#) () const
- **CertificateInfo** [subjectInfo](#) () const
- [CertificateInfoOrdered](#) [subjectInfoOrdered](#) () const
- **CertificateInfo** [issuerInfo](#) () const
- [CertificateInfoOrdered](#) [issuerInfoOrdered](#) () const
- **Constraints** [constraints](#) () const
- **QStringList** [policies](#) () const
- **QStringList** [crlLocations](#) () const
- **QStringList** [issuerLocations](#) () const
- **QStringList** [ocspLocations](#) () const
- **QString** [commonName](#) () const
- **BigInteger** [serialNumber](#) () const
- **PublicKey** [subjectPublicKey](#) () const
- bool [isCA](#) () const
- bool [isSelfSigned](#) () const

- bool `isIssuerOf` (const [Certificate](#) &other) const
- int `pathLimit` () const
- [SignatureAlgorithm](#) `signatureAlgorithm` () const
- [QByteArray](#) `subjectKeyId` () const
- [QByteArray](#) `issuerKeyId` () const
- [Validity](#) `validate` (const [CertificateCollection](#) &trusted, const [CertificateCollection](#) &untrusted, [UsageMode](#) u=UsageAny, [ValidateFlags](#) vf=ValidateAll) const
- [QByteArray](#) `toDER` () const
- [QString](#) `toPEM` () const
- bool `toPEMFile` (const [QString](#) &fileName) const
- bool `matchesHostName` (const [QString](#) &host) const
- bool `operator==` (const [Certificate](#) &a) const
- bool `operator!=` (const [Certificate](#) &other) const
- void `change` (CertContext *c)

Static Public Member Functions

- static [Certificate](#) `fromDER` (const [QByteArray](#) &a, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [Certificate](#) `fromPEM` (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [Certificate](#) `fromPEMFile` (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

Friends

- class [Private](#)
- class [CertificateChain](#)

10.7.1 Detailed Description

Public Key (X.509) certificate.

This class contains one X.509 certificate

Examples:

[certtest.cpp](#), [cmsexample.cpp](#), [publickeyexample.cpp](#), [sslservtest.cpp](#), and [ssltest.cpp](#).

10.7.2 Constructor & Destructor Documentation

10.7.2.1 [QCA::Certificate::Certificate](#) ()

Create an empty [Certificate](#).

10.7.2.2 [QCA::Certificate::Certificate](#) (const [QString](#) &*fileName*)

Create a [Certificate](#) from a PEM encoded file.

Parameters:

fileName the name (and path, if required) of the file that contains the PEM encoded certificate

10.7.2.3 QCA::Certificate::Certificate (const [CertificateOptions](#) & *opts*, const [PrivateKey](#) & *key*, const QString & *provider* = QString ())

Create a [Certificate](#) with specified options and a specified private key.

Parameters:

opts the options to use

key the private key for this certificate

provider the provider to use to create this key, if a particular provider is required

10.7.2.4 QCA::Certificate::Certificate (const [Certificate](#) & *from*)

Standard copy constructor.

10.7.3 Member Function Documentation

10.7.3.1 [Certificate](#)& QCA::Certificate::operator= (const [Certificate](#) & *from*)

Standard assignment operator.

10.7.3.2 bool QCA::Certificate::isNull () const

Test if the certificate is empty (null).

Returns:

true if the certificate is null

Examples:

[cmsexample.cpp](#), [publickeyexample.cpp](#), [sslservtest.cpp](#), and [ssltest.cpp](#).

10.7.3.3 QDateTime QCA::Certificate::notValidBefore () const

The earliest date that the certificate is valid.

Examples:

[certtest.cpp](#), and [ssltest.cpp](#).

10.7.3.4 QDateTime QCA::Certificate::notValidAfter () const

The latest date that the certificate is valid.

Examples:

[certtest.cpp](#), and [ssltest.cpp](#).

10.7.3.5 `CertificateInfo QCA::Certificate::subjectInfo () const`

Properties of the subject of the certificate, as a `QMultiMap`.

This is the method that provides information on the subject organisation, common name, DNS name, and so on. The list of information types (i.e. the key to the multi-map) is a `CertificateInfoType`. The values are a list of `QString`.

An example of how you can iterate over the list is:

```
foreach( QString dns, info.values(QCA::DNS) )
{
    std::cout << "    " << qPrintable(dns) << std::endl;
}
```

Examples:

[certtest.cpp](#).

10.7.3.6 `CertificateInfoOrdered QCA::Certificate::subjectInfoOrdered () const`

Properties of the subject of the certificate, as an ordered list (`QList` of `CertificateInfoPair`).

This allows access to the certificate information in the same order as they appear in a certificate. Each pair in the list has a type and a value.

For example:

```
CertificateInfoOrdered info = cert.subjectInfoOrdered();
// info[0].type == CommonName
// info[0].value == "example.com"
```

See also:

[subjectInfo](#) for an unordered version
[issuerInfoOrdered](#) for the ordered information on the issuer
[CertificateInfoPair](#) for the elements in the list

10.7.3.7 `CertificateInfo QCA::Certificate::issuerInfo () const`

Properties of the issuer of the certificate.

See also:

[subjectInfo](#) for how the return value works.

Examples:

[certtest.cpp](#).

10.7.3.8 `CertificateInfoOrdered QCA::Certificate::issuerInfoOrdered () const`

Properties of the issuer of the certificate, as an ordered list (`QList` of `CertificateInfoPair`).

This allows access to the certificate information in the same order as they appear in a certificate. Each pair in the list has a type and a value.

See also:

[issuerInfo](#) for an unordered version
[subjectInfoOrdered](#) for the ordered information on the subject
[CertificateInfoPair](#) for the elements in the list

10.7.3.9 Constraints QCA::Certificate::constraints () const

The constraints that apply to this certificate.

10.7.3.10 QStringList QCA::Certificate::policies () const

The policies that apply to this certificate.

Policies are specified as strings containing OIDs

10.7.3.11 QStringList QCA::Certificate::crlLocations () const

list of URI locations for [CRL](#) files

each URI refers to the same [CRL](#) file

10.7.3.12 QStringList QCA::Certificate::issuerLocations () const

list of URI locations for issuer certificate files

each URI refers to the same issuer file

10.7.3.13 QStringList QCA::Certificate::ocspLocations () const

list of URI locations for OCSP services

10.7.3.14 QString QCA::Certificate::commonName () const

The common name of the subject of the certificate.

Common names are normally the name of a person, company or organisation

Examples:

[ssltest.cpp](#).

10.7.3.15 BigInteger QCA::Certificate::serialNumber () const

The serial number of the certificate.

Examples:

[certtest.cpp](#).

10.7.3.16 [PublicKey](#) QCA::Certificate::subjectPublicKey () const

The public key associated with the subject of the certificate.

10.7.3.17 `bool` QCA::Certificate::isCA () const

Test if the [Certificate](#) is valid as a [Certificate](#) Authority.

Returns:

true if the [Certificate](#) is valid as a [Certificate](#) Authority

Examples:

[certtest.cpp](#).

10.7.3.18 `bool` QCA::Certificate::isSelfSigned () const

Test if the [Certificate](#) is self-signed.

Returns:

true if the certificate is self-signed

Examples:

[certtest.cpp](#).

10.7.3.19 `bool` QCA::Certificate::isIssuerOf (const [Certificate](#) & *other*) const

Test if the [Certificate](#) has signed another [Certificate](#) object and is therefore the issuer.

Returns:

true if the certificate is the issuer

10.7.3.20 `int` QCA::Certificate::pathLimit () const

The upper bound of the number of links in the certificate chain, if any.

10.7.3.21 [SignatureAlgorithm](#) QCA::Certificate::signatureAlgorithm () const

The signature algorithm used for the signature on this certificate.

10.7.3.22 `QByteArray` QCA::Certificate::subjectKeyId () const

The key identifier associated with the subject.

10.7.3.23 `QByteArray` QCA::Certificate::issuerKeyId () const

The key identifier associated with the issuer.

10.7.3.24 **Validity** `QCA::Certificate::validate (const CertificateCollection & trusted, const CertificateCollection & untrusted, UsageMode u = UsageAny, ValidateFlags vf = ValidateAll) const`

Check the validity of a certificate.

Parameters:

- trusted* a collection of trusted certificates
- untrusted* a collection of additional certificates, not necessarily trusted
- u* the use required for the certificate
- vf* the conditions to validate

Note:

This function may block

10.7.3.25 `QByteArray QCA::Certificate::toDER () const`

Export the [Certificate](#) into a DER format.

10.7.3.26 `QString QCA::Certificate::toPEM () const`

Export the [Certificate](#) into a PEM format.

Examples:

[certtest.cpp](#), and [ssltest.cpp](#).

10.7.3.27 `bool QCA::Certificate::toPEMFile (const QString & fileName) const`

Export the [Certificate](#) into PEM format in a file.

Parameters:

- fileName* the name of the file to use

10.7.3.28 `static Certificate QCA::Certificate::fromDER (const QByteArray & a, ConvertResult * result = 0, const QString & provider = QString ()) [static]`

Import the certificate from DER.

Parameters:

- a* the array containing the certificate in DER format
- result* a pointer to a [ConvertResult](#), which if not-null will be set to the conversion status
- provider* the provider to use, if a specific provider is required

Returns:

the [Certificate](#) corresponding to the certificate in the provided array

10.7.3.29 static [Certificate](#) QCA::Certificate::fromPEM (const QString & *s*, [ConvertResult](#) * *result* = 0, const QString & *provider* = QString()) [static]

Import the certificate from PEM format.

Parameters:

s the string containing the certificate in PEM format

result a pointer to a ConvertResult, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [Certificate](#) corresponding to the certificate in the provided string

Examples:

[sslservtest.cpp](#), and [ssltest.cpp](#).

10.7.3.30 static [Certificate](#) QCA::Certificate::fromPEMFile (const QString & *fileName*, [ConvertResult](#) * *result* = 0, const QString & *provider* = QString()) [static]

Import the certificate from a file.

Parameters:

fileName the name (and path, if required) of the file containing the certificate in PEM format

result a pointer to a ConvertResult, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [Certificate](#) corresponding to the certificate in the provided string

10.7.3.31 bool QCA::Certificate::matchesHostName (const QString & *host*) const

Test if the subject of the certificate matches a specified host name.

This will return true (indicating a match), if the specified host name meets the RFC 2818 validation rules with this certificate.

If the host is an internationalized domain name, then it must be provided in unicode format, not in IDNA ACE/punycode format.

Parameters:

host the name of the host to compare to

10.7.3.32 bool QCA::Certificate::operator== (const [Certificate](#) & *a*) const

Test for equality of two certificates.

Returns:

true if the two certificates are the same

10.7.3.33 `bool QCA::Certificate::operator!= (const Certificate & other) const` `[inline]`

Inequality operator.

10.7.3.34 `void QCA::Certificate::change (CertContext * c)`

For internal use only.

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

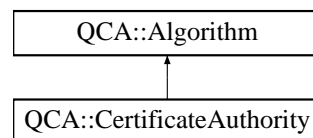
- [qca_cert.h](#)

10.8 QCA::CertificateAuthority Class Reference

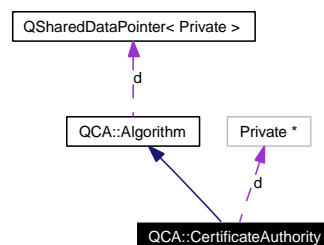
A Certificate Authority is used to generate Certificates and Certificate Revocation Lists (CRLs).

```
#include <QtCrypto>
```

Inheritance diagram for QCA::CertificateAuthority::



Collaboration diagram for QCA::CertificateAuthority:



Public Member Functions

- [CertificateAuthority](#) (const [Certificate](#) &cert, const [PrivateKey](#) &key, const [QString](#) &provider)
- [CertificateAuthority](#) (const [CertificateAuthority](#) &from)
- [CertificateAuthority](#) & operator= (const [CertificateAuthority](#) &from)
- [Certificate](#) [certificate](#) () const
- [Certificate](#) [signRequest](#) (const [CertificateRequest](#) &req, const [QDateTime](#) ¬ValidAfter) const
- [Certificate](#) [createCertificate](#) (const [PublicKey](#) &key, const [CertificateOptions](#) &opts) const
- [CRL](#) [createCRL](#) (const [QDateTime](#) &nextUpdate) const
- [CRL](#) [updateCRL](#) (const [CRL](#) &crl, const [QList](#)< [CRLEntry](#) > &entries, const [QDateTime](#) &nextUpdate) const

10.8.1 Detailed Description

A Certificate Authority is used to generate Certificates and Certificate Revocation Lists (CRLs).

10.8.2 Constructor & Destructor Documentation

10.8.2.1 QCA::CertificateAuthority::CertificateAuthority (const [Certificate](#) & cert, const [PrivateKey](#) & key, const [QString](#) & provider)

Create a new Certificate Authority.

Parameters:

cert the CA certificate

key the private key associated with the CA certificate
provider the provider to use, if a specific provider is required

10.8.2.2 QCA::CertificateAuthority::CertificateAuthority (const [CertificateAuthority](#) & *from*)

Copy constructor.

Parameters:

from the [CertificateAuthority](#) to copy from

10.8.3 Member Function Documentation

10.8.3.1 [CertificateAuthority](#) & QCA::CertificateAuthority::operator= (const [CertificateAuthority](#) & *from*)

Standard assignment operator.

Parameters:

from the [CertificateAuthority](#) to copy from

10.8.3.2 [Certificate](#) QCA::CertificateAuthority::certificate () const

The [Certificate](#) belonging to the CertificateAuthority.

This is the [Certificate](#) that was passed as an argument to the constructor

10.8.3.3 [Certificate](#) QCA::CertificateAuthority::signRequest (const [CertificateRequest](#) & *req*, const QDateTime & *notValidAfter*) const

Create a new [Certificate](#) by signing the provider [CertificateRequest](#).

Parameters:

req the [CertificateRequest](#) to sign

notValidAfter the last date that the [Certificate](#) will be valid

10.8.3.4 [Certificate](#) QCA::CertificateAuthority::createCertificate (const [PublicKey](#) & *key*, const [CertificateOptions](#) & *opts*) const

Create a new [Certificate](#).

Parameters:

key the Public Key to use to create the [Certificate](#)

opts the options to use for the new [Certificate](#)

10.8.3.5 CRL QCA::CertificateAuthority::createCRL (const QDateTime & *nextUpdate*) const

Create a new [Certificate](#) Revocation List ([CRL](#)).

Parameters:

nextUpdate the date that the [CRL](#) will be updated

Returns:

an empty [CRL](#)

10.8.3.6 CRL QCA::CertificateAuthority::updateCRL (const [CRL](#) & *crl*, const QList< [CRL](#)Entry > & *entries*, const QDateTime & *nextUpdate*) const

Update the [CRL](#) to include new entries.

Parameters:

crl the [CRL](#) to update

entries the entries to add to the [CRL](#)

nextUpdate the date that this [CRL](#) will be updated

Returns:

the update [CRL](#)

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

- [qca_cert.h](#)

10.9 QCA::CertificateChain Class Reference

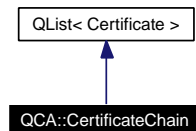
A chain of related Certificates.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::CertificateChain::



Collaboration diagram for QCA::CertificateChain:



Public Member Functions

- [CertificateChain](#) ()
- [CertificateChain](#) (const [Certificate](#) &primary)
- const [Certificate](#) & [primary](#) () const
- [Validity](#) [validate](#) (const [CertificateCollection](#) &trusted, const [QList](#)< [CRL](#) > &untrusted_
crls=[QList](#)< [CRL](#) >(), [UsageMode](#) u=UsageAny, [ValidateFlags](#) vf=ValidateAll) const
- [CertificateChain](#) [complete](#) (const [QList](#)< [Certificate](#) > &issuers=[QList](#)< [Certificate](#) >(), [Validity](#)
*result=0) const

10.9.1 Detailed Description

A chain of related Certificates.

[CertificateChain](#) is a list (a [QList](#)) of certificates that are related by the signature from one to another. If [Certificate](#) C signs [Certificate](#) B, and [Certificate](#) B signs [Certificate](#) A, then C, B and A form a chain.

The normal use of a [CertificateChain](#) is from a end-user [Certificate](#) (called the primary, equivalent to [QList::first\(\)](#)) through some intermediate Certificates to some other [Certificate](#) ([QList::last\(\)](#)), which might be a root [Certificate](#) Authority, but does not need to be.

You can build up the chain using normal [QList](#) operations, such as [QList::append\(\)](#).

See also:

[QCA::CertificateCollection](#) for an alternative way to represent a group of Certificates that do not necessarily have a chained relationship.

Examples:

[cmsexample.cpp](#), and [publickeyexample.cpp](#).

10.9.2 Constructor & Destructor Documentation

10.9.2.1 QCA::CertificateChain::CertificateChain () [inline]

Create an empty certificate chain.

10.9.2.2 QCA::CertificateChain::CertificateChain (const [Certificate](#) & *primary*) [inline]

Create a certificate chain, starting at the specified certificate.

Parameters:

primary the end-user certificate that forms one end of the chain

10.9.3 Member Function Documentation

10.9.3.1 const [Certificate](#)& QCA::CertificateChain::primary () const [inline]

Return the primary (end-user) [Certificate](#).

10.9.3.2 **Validity** QCA::CertificateChain::validate (const [CertificateCollection](#) & *trusted*, const [QList](#)< [CRL](#) > & *untrusted_crls* = [QList](#)< [CRL](#) > (), [UsageMode](#) *u* = UsageAny, [ValidateFlags](#) *vf* = ValidateAll) const [inline]

Check the validity of a certificate chain.

Parameters:

trusted a collection of trusted certificates

untrusted_crls a list of additional CRLs, not necessarily trusted

u the use required for the primary certificate

vf the conditions to validate

Note:

This function may block

See also:

[Certificate::validate\(\)](#)

10.9.3.3 [CertificateChain](#) QCA::CertificateChain::complete (const [QList](#)< [Certificate](#) > & *issuers* = [QList](#)< [Certificate](#) > (), [Validity](#) * *result* = 0) const [inline]

Complete a certificate chain for the primary certificate, using the rest of the certificates in the chain object, as well as those in *issuers*, as possible issuers in the chain.

If there are issuers missing, then the chain might be incomplete (at the worst case, if no issuers exist for the primary certificate, then the resulting chain will consist of just the primary certificate). Use the *result* argument to find out if there was a problem during completion. A result of [ValidityGood](#) means the chain was completed successfully.

The newly constructed [CertificateChain](#) is returned.

If the certificate chain is empty, then this will return an empty [CertificateChain](#) object.

Parameters:

issuers a pool of issuers to draw from as necessary
result the result of the completion operation

Note:

This function may block

See also:

[validate](#)

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

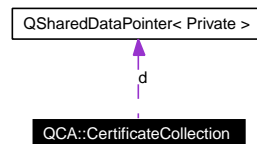
- [qca_cert.h](#)

10.10 QCA::CertificateCollection Class Reference

Bundle of Certificates and CRLs.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::CertificateCollection:



Public Member Functions

- [CertificateCollection](#) ()
- [CertificateCollection](#) (const [CertificateCollection](#) &from)
- [CertificateCollection](#) & operator= (const [CertificateCollection](#) &from)
- void [addCertificate](#) (const [Certificate](#) &cert)
- void [addCRL](#) (const [CRL](#) &crl)
- [QList](#)< [Certificate](#) > [certificates](#) () const
- [QList](#)< [CRL](#) > [crls](#) () const
- void [append](#) (const [CertificateCollection](#) &other)
- [CertificateCollection](#) operator+ (const [CertificateCollection](#) &other) const
- [CertificateCollection](#) & operator+= (const [CertificateCollection](#) &other)
- bool [toFlatTextFile](#) (const [QString](#) &fileName)
- bool [toPKCS7File](#) (const [QString](#) &fileName, const [QString](#) &provider=[QString](#)())

Static Public Member Functions

- static bool [canUsePKCS7](#) (const [QString](#) &provider=[QString](#)())
- static [CertificateCollection](#) [fromFlatTextFile](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [CertificateCollection](#) [fromPKCS7File](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

10.10.1 Detailed Description

Bundle of Certificates and CRLs.

[CertificateCollection](#) provides a bundle of Certificates and [Certificate](#) Revocation Lists (CRLs), not necessarily related.

See also:

[QCA::CertificateChain](#) for a representation of a chain of Certificates related by signatures.

Examples:

[certtest.cpp](#), and [ssltest.cpp](#).

10.10.2 Constructor & Destructor Documentation

10.10.2.1 QCA::CertificateCollection::CertificateCollection ()

Create an empty [Certificate](#) / [CRL](#) collection.

10.10.2.2 QCA::CertificateCollection::CertificateCollection (const [CertificateCollection](#) & *from*)

Standard copy constructor.

Parameters:

from the [CertificateCollection](#) to copy from

10.10.3 Member Function Documentation

10.10.3.1 [CertificateCollection](#)& QCA::CertificateCollection::operator= (const [CertificateCollection](#) & *from*)

Standard assignment operator.

Parameters:

from the [CertificateCollection](#) to copy from

10.10.3.2 void QCA::CertificateCollection::addCertificate (const [Certificate](#) & *cert*)

Append a [Certificate](#) to this collection.

Parameters:

cert the [Certificate](#) to add to this [CertificateCollection](#)

Examples:

[ssltest.cpp](#).

10.10.3.3 void QCA::CertificateCollection::addCRL (const [CRL](#) & *crl*)

Append a [CRL](#) to this collection.

Parameters:

crl the certificate revocation list to add to this [CertificateCollection](#)

10.10.3.4 QList<[Certificate](#)> QCA::CertificateCollection::certificates () const

The Certificates in this collection.

Examples:

[certtest.cpp](#).

10.10.3.5 `QList<CRL> QCA::CertificateCollection::crls () const`

The CRLs in this collection.

10.10.3.6 `void QCA::CertificateCollection::append (const CertificateCollection & other)`

Add another [CertificateCollection](#) to this collection.

Parameters:

other the [CertificateCollection](#) to add to this collection

10.10.3.7 `CertificateCollection QCA::CertificateCollection::operator+ (const CertificateCollection & other) const`

Add another [CertificateCollection](#) to this collection.

Parameters:

other the [CertificateCollection](#) to add to this collection

10.10.3.8 `CertificateCollection& QCA::CertificateCollection::operator+= (const CertificateCollection & other)`

Add another [CertificateCollection](#) to this collection.

Parameters:

other the [CertificateCollection](#) to add to this collection

10.10.3.9 `static bool QCA::CertificateCollection::canUsePKCS7 (const QString & provider = QString()) [static]`

test if the [CertificateCollection](#) can be imported and exported to PKCS#7 format

Parameters:

provider the provider to use, if a specific provider is required

Returns:

true if the [CertificateCollection](#) can be imported and exported to PKCS#7 format

10.10.3.10 `bool QCA::CertificateCollection::toFlatTextFile (const QString & fileName)`

export the [CertificateCollection](#) to a plain text file

Parameters:

fileName the name (and path, if required) to write the contents of the [CertificateCollection](#) to

Returns:

true if the export succeeded, otherwise false

10.10.3.11 `bool QCA::CertificateCollection::toPKCS7File (const QString & fileName, const QString & provider = QString ())`

export the [CertificateCollection](#) to a PKCS#7 file

Parameters:

fileName the name (and path, if required) to write the contents of the [CertificateCollection](#) to

provider the provider to use, if a specific provider is required

Returns:

true if the export succeeded, otherwise false

10.10.3.12 `static CertificateCollection QCA::CertificateCollection::fromFlatTextFile (const QString & fileName, ConvertResult * result = 0, const QString & provider = QString ()) [static]`

import a [CertificateCollection](#) from a text file

Parameters:

fileName the name (and path, if required) to read the certificate collection from

result a pointer to a [ConvertResult](#), which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CertificateCollection](#) corresponding to the contents of the file specified in *fileName*

Examples:

[certtest.cpp](#).

10.10.3.13 `static CertificateCollection QCA::CertificateCollection::fromPKCS7File (const QString & fileName, ConvertResult * result = 0, const QString & provider = QString ()) [static]`

import a [CertificateCollection](#) from a PKCS#7 file

Parameters:

fileName the name (and path, if required) to read the certificate collection from

result a pointer to a [ConvertResult](#), which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CertificateCollection](#) corresponding to the contents of the file specified in *fileName*

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

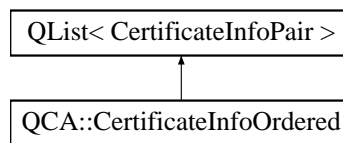
- [qca_cert.h](#)

10.11 QCA::CertificateInfoOrdered Class Reference

Ordered certificate properties type.

```
#include <qca_cert.h>
```

Inheritance diagram for QCA::CertificateInfoOrdered::



Collaboration diagram for QCA::CertificateInfoOrdered:



Public Member Functions

- [QString toString \(\) const](#)
- [CertificateInfoOrdered dnOnly \(\) const](#)

10.11.1 Detailed Description

Ordered certificate properties type.

This container stores the information in the same sequence as the certificate format itself.

Examples:

[ssltest.cpp](#).

10.11.2 Member Function Documentation

10.11.2.1 QString QCA::CertificateInfoOrdered::toString () const [inline]

Convert to RFC 1779 string format.

10.11.2.2 [CertificateInfoOrdered](#) QCA::CertificateInfoOrdered::dnOnly () const [inline]

Return a new [CertificateInfoOrdered](#) that only contains the Distinguished Name (DN) types found in this object.

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

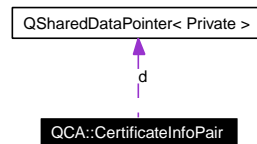
- [qca_cert.h](#)

10.12 QCA::CertificateInfoPair Class Reference

One entry in a certificate information list.

```
#include <qca_cert.h>
```

Collaboration diagram for QCA::CertificateInfoPair:



Public Member Functions

- [CertificateInfoPair](#) ()
- [CertificateInfoPair](#) (const [CertificateInfoType](#) &type, const **QString** &value)
- [CertificateInfoPair](#) (const [CertificateInfoPair](#) &from)
- [CertificateInfoPair](#) & operator= (const [CertificateInfoPair](#) &from)
- [CertificateInfoType](#) type () const
- **QString** value () const
- bool operator== (const [CertificateInfoPair](#) &other) const
- bool operator!= (const [CertificateInfoPair](#) &other) const

10.12.1 Detailed Description

One entry in a certificate information list.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 QCA::CertificateInfoPair::CertificateInfoPair ()

Standard constructor.

10.12.2.2 QCA::CertificateInfoPair::CertificateInfoPair (const [CertificateInfoType](#) &type, const **QString** &value)

Construct a new pair.

Parameters:

- type* the type of information stored in this pair
- value* the value of the information to be stored

10.12.2.3 QCA::CertificateInfoPair::CertificateInfoPair (const [CertificateInfoPair](#) &from)

Standard copy constructor.

10.12.3 Member Function Documentation

10.12.3.1 [CertificateInfoPair](#)& QCA::CertificateInfoPair::operator= (const [CertificateInfoPair](#) & *from*)

Standard assignment operator.

10.12.3.2 [CertificateInfoType](#) QCA::CertificateInfoPair::type () const

The type of information stored in the pair.

10.12.3.3 QString QCA::CertificateInfoPair::value () const

The value of the information stored in the pair.

10.12.3.4 bool QCA::CertificateInfoPair::operator== (const [CertificateInfoPair](#) & *other*) const

Comparison operator.

10.12.3.5 bool QCA::CertificateInfoPair::operator!= (const [CertificateInfoPair](#) & *other*) const [inline]

Inequality operator.

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

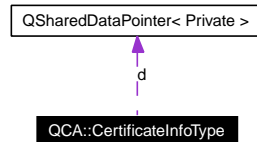
- [qca_cert.h](#)

10.13 QCA::CertificateInfoType Class Reference

Certificate information type.

```
#include <qca_cert.h>
```

Collaboration diagram for QCA::CertificateInfoType:



Public Types

- enum **Section** { **DN**, **AlternativeName** }

Public Member Functions

- **CertificateInfoType** ()
- **CertificateInfoType** (**CertificateInfoTypeKnown** known)
- **CertificateInfoType** (const **QString** &id, **Section** section)
- **CertificateInfoType** (const **CertificateInfoType** &from)
- **CertificateInfoType** & **operator=** (const **CertificateInfoType** &from)
- **Section** section () const
- **CertificateInfoTypeKnown** known () const
- **QString** id () const
- bool **operator<** (const **CertificateInfoType** &other) const
- bool **operator==** (const **CertificateInfoType** &other) const
- bool **operator!=** (const **CertificateInfoType** &other) const

10.13.1 Detailed Description

Certificate information type.

This class represents a type of information being stored in a certificate. It can be created either using a known type (from the Known enumerator) or an identifier string (usually an OID). Types created either way are interchangeable.

Types also have the notion of a Section. Some types may reside in the Distinguished Name field of a certificate, and some types may reside in the Subject Alternative Name field. This class is capable of representing a type from either section.

In the general case, applications will want to use the CertificateInfoTypeKnown enumerator types. These are from RFC3280 (<http://www.ietf.org/rfc/rfc3280.txt>) except where shown.

The entries for IncorporationLocality, IncorporationState and IncorporationCountry are the same as Locality, State and Country respectively, except that the Extended Validation (EV) certificate guidelines (published by the Certificate Authority / Browser Forum, see <http://www.cabforum.org>) distinguish between the place of where the company does business (which is the Locality / State / Country combination) and the jurisdiction where the company is legally incorporated (the IncorporationLocality / IncorporationState / IncorporationCountry combination).

See also:

[Certificate::subjectInfo\(\)](#) and [Certificate::issuerInfo\(\)](#)
[CRL::issuerInfo\(\)](#)

10.13.2 Member Enumeration Documentation

10.13.2.1 enum [QCA::CertificateInfoType::Section](#)

Section of the certificate that the information belongs in.

Enumerator:

DN Distinguished name (the primary name).

AlternativeName Alternative name.

10.13.3 Constructor & Destructor Documentation

10.13.3.1 [QCA::CertificateInfoType::CertificateInfoType \(\)](#)

Standard constructor.

10.13.3.2 [QCA::CertificateInfoType::CertificateInfoType \(\[CertificateInfoTypeKnown\]\(#\) *known*\)](#)

Construct a new type.

The section will be derived by *known*.

Parameters:

known the type as part of the CertificateInfoTypeKnown enumerator

10.13.3.3 [QCA::CertificateInfoType::CertificateInfoType \(const QString & *id*, \[Section\]\(#\) *section*\)](#)

Construct a new type.

Parameters:

id the type as an identifier string (OID or internal)

section the section this type belongs in

See also:

[id](#)

10.13.3.4 [QCA::CertificateInfoType::CertificateInfoType \(const \[CertificateInfoType\]\(#\) & *from*\)](#)

Standard copy constructor.

10.13.4 Member Function Documentation

10.13.4.1 [CertificateInfoType&](#) [QCA::CertificateInfoType::operator= \(const \[CertificateInfoType\]\(#\) & *from*\)](#)

Standard assignment operator.

10.13.4.2 [Section](#) QCA::CertificateInfoType::section () const

The section the type is part of.

10.13.4.3 [CertificateInfoTypeKnown](#) QCA::CertificateInfoType::known () const

The type as part of the CertificateInfoTypeKnown enumerator.

This function may return a value that does not exist in the enumerator. In that case, you may use [id\(\)](#) to determine the type.

10.13.4.4 [QString](#) QCA::CertificateInfoType::id () const

The type as an identifier string.

For types that have OIDs, this function returns an OID in string form. For types that do not have OIDs, this function returns an internal identifier string whose first character is not a digit (this allows you to tell the difference between an OID and an internal identifier).

It is hereby stated that General Names (of the X.509 Subject Alternative Name) shall use the internal identifier format "GeneralName.[rfc field name]". For example, the rfc822Name field would have the identifier "GeneralName.rfc822Name".

Applications should not store, use, or compare against internal identifiers unless the identifiers are explicitly documented (e.g. GeneralName).

10.13.4.5 [bool](#) QCA::CertificateInfoType::operator< (const [CertificateInfoType](#) & *other*) const

Comparison operator.

10.13.4.6 [bool](#) QCA::CertificateInfoType::operator== (const [CertificateInfoType](#) & *other*) const

Comparison operator.

10.13.4.7 [bool](#) QCA::CertificateInfoType::operator!= (const [CertificateInfoType](#) & *other*) const [inline]

Inequality operator.

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

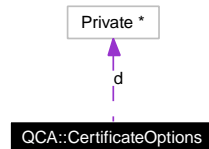
- [qca_cert.h](#)

10.14 QCA::CertificateOptions Class Reference

Certificate options

```
#include <QtCrypto>
```

Collaboration diagram for QCA::CertificateOptions:



Public Member Functions

- [CertificateOptions](#) ([CertificateRequestFormat](#) format=PKCS10)
- [CertificateOptions](#) (const [CertificateOptions](#) &from)
- [CertificateOptions](#) & [operator=](#) (const [CertificateOptions](#) &from)
- [CertificateRequestFormat](#) [format](#) () const
- void [setFormat](#) ([CertificateRequestFormat](#) f)
- bool [isValid](#) () const
- [QString](#) [challenge](#) () const
- [CertificateInfo](#) [info](#) () const
- [CertificateInfoOrdered](#) [infoOrdered](#) () const
- [Constraints](#) [constraints](#) () const
- [QStringList](#) [policies](#) () const
- [QStringList](#) [crlLocations](#) () const
- [QStringList](#) [issuerLocations](#) () const
- [QStringList](#) [ocspLocations](#) () const
- bool [isCA](#) () const
- int [pathLimit](#) () const
- [BigInteger](#) [serialNumber](#) () const
- [QDateTime](#) [notValidBefore](#) () const
- [QDateTime](#) [notValidAfter](#) () const
- void [setChallenge](#) (const [QString](#) &s)
- void [setInfo](#) (const [CertificateInfo](#) &info)
- void [setInfoOrdered](#) (const [CertificateInfoOrdered](#) &info)
- void [setConstraints](#) (const [Constraints](#) &constraints)
- void [setPolicies](#) (const [QStringList](#) &policies)
- void [setCRLLocations](#) (const [QStringList](#) &locations)
- void [setIssuerLocations](#) (const [QStringList](#) &locations)
- void [setOCSPLocations](#) (const [QStringList](#) &locations)
- void [setAsCA](#) (int pathLimit=8)
- void [setAsUser](#) ()
- void [setSerialNumber](#) (const [BigInteger](#) &i)
- void [setValidityPeriod](#) (const [QDateTime](#) &start, const [QDateTime](#) &end)

10.14.1 Detailed Description

Certificate options

Note:

In SPKAC mode, all options are ignored except for challenge

10.14.2 Constructor & Destructor Documentation

10.14.2.1 QCA::CertificateOptions::CertificateOptions ([CertificateRequestFormat](#) *format* = PKCS10)

Create a [Certificate](#) options set.

Parameters:

format the format to create the certificate request in

10.14.2.2 QCA::CertificateOptions::CertificateOptions (const [CertificateOptions](#) & *from*)

Standard copy constructor.

Parameters:

from the [Certificate](#) Options to copy into this object

10.14.3 Member Function Documentation

10.14.3.1 [CertificateOptions](#)& QCA::CertificateOptions::operator= (const [CertificateOptions](#) & *from*)

Standard assignment operator.

Parameters:

from the [Certificate](#) Options to copy into this object

10.14.3.2 [CertificateRequestFormat](#) QCA::CertificateOptions::format () const

test the format type for this certificate

10.14.3.3 void QCA::CertificateOptions::setFormat ([CertificateRequestFormat](#) *f*)

Specify the format for this certificate.

Parameters:

f the format to use

10.14.3.4 bool QCA::CertificateOptions::isValid () const

Test if the certificate options object is valid.

Returns:

true if the certificate options object is valid

10.14.3.5 QString QCA::CertificateOptions::challenge () const

The challenge part of the certificate.

For [CertificateRequest](#) only

See also:

[setChallenge](#)

10.14.3.6 CertificateInfo QCA::CertificateOptions::info () const

Information on the subject of the certificate.

See also:

[setInfo](#)

10.14.3.7 [CertificateInfoOrdered](#) QCA::CertificateOptions::infoOrdered () const

Information on the subject of the certificate, in the exact order the items will be written.

See also:

[setInfoOrdered](#)

10.14.3.8 Constraints QCA::CertificateOptions::constraints () const

List the constraints on this certificate.

10.14.3.9 QStringList QCA::CertificateOptions::policies () const

list the policies on this certificate

10.14.3.10 QStringList QCA::CertificateOptions::crlLocations () const

list of URI locations for [CRL](#) files

each URI refers to the same [CRL](#) file

For [Certificate](#) creation only

10.14.3.11 QStringList QCA::CertificateOptions::issuerLocations () const

list of URI locations for issuer certificate files

each URI refers to the same issuer file

For [Certificate](#) creation only

10.14.3.12 QStringList QCA::CertificateOptions::ocspLocations () const

list of URI locations for OCSP services

For [Certificate](#) creation only

10.14.3.13 bool QCA::CertificateOptions::isCA () const

test if the certificate is a CA cert

See also:

[setAsCA](#)

[setAsUser](#)

10.14.3.14 int QCA::CertificateOptions::pathLimit () const

return the path limit on this certificate

10.14.3.15 [BigInteger](#) QCA::CertificateOptions::serialNumber () const

The serial number for the certificate.

For [Certificate](#) creation only

10.14.3.16 QDateTime QCA::CertificateOptions::notValidBefore () const

the first time the certificate will be valid

For [Certificate](#) creation only

10.14.3.17 QDateTime QCA::CertificateOptions::notValidAfter () const

the last time the certificate is valid

For [Certificate](#) creation only

10.14.3.18 void QCA::CertificateOptions::setChallenge (const QString & s)

Specify the challenge associated with this certificate.

Parameters:

s the challenge string

See also:

[challenge\(\)](#)

10.14.3.19 void QCA::CertificateOptions::setInfo (const CertificateInfo & *info*)

Specify information for the the subject associated with the certificate.

Parameters:

info the information for the subject

See also:

[info\(\)](#)

10.14.3.20 void QCA::CertificateOptions::setInfoOrdered (const CertificateInfoOrdered & *info*)

Specify information for the the subject associated with the certificate.

Parameters:

info the information for the subject

See also:

[info\(\)](#)

10.14.3.21 void QCA::CertificateOptions::setConstraints (const Constraints & *constraints*)

set the constraints on the certificate

Parameters:

constraints the constraints to be used for the certificate

10.14.3.22 void QCA::CertificateOptions::setPolicies (const QStringList & *policies*)

set the policies on the certificate

Parameters:

policies the policies to be used for the certificate

10.14.3.23 void QCA::CertificateOptions::setCRLLocations (const QStringList & *locations*)

set the [CRL](#) locations of the certificate

each location refers to the same [CRL](#).

Parameters:

locations a list of URIs to [CRL](#) files

10.14.3.24 void QCA::CertificateOptions::setIssuerLocations (const QStringList & *locations*)

set the issuer certificate locations of the certificate
each location refers to the same issuer file.

Parameters:

locations a list of URIs to issuer certificate files

10.14.3.25 void QCA::CertificateOptions::setOCSPLocations (const QStringList & *locations*)

set the OCSP service locations of the certificate

Parameters:

locations a list of URIs to OCSP services

10.14.3.26 void QCA::CertificateOptions::setAsCA (int *pathLimit* = 8)

set the certificate to be a CA cert

Parameters:

pathLimit the number of intermediate certificates allowable

10.14.3.27 void QCA::CertificateOptions::setAsUser ()

set the certificate to be a user cert (this is the default)

10.14.3.28 void QCA::CertificateOptions::setSerialNumber (const [BigInteger](#) & *i*)

Set the serial number property on this certificate.

Parameters:

i the serial number to use

10.14.3.29 void QCA::CertificateOptions::setValidityPeriod (const QDateTime & *start*, const QDateTime & *end*)

Set the validity period for the certificate.

Parameters:

start the first time this certificate becomes valid

end the last time this certificate is valid

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

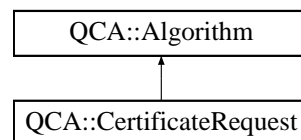
- [qca_cert.h](#)

10.15 QCA::CertificateRequest Class Reference

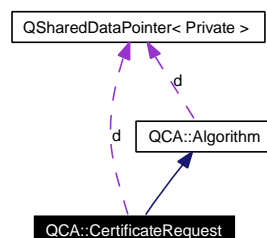
Certificate Request

```
#include <QtCrypto>
```

Inheritance diagram for QCA::CertificateRequest:



Collaboration diagram for QCA::CertificateRequest:



Public Member Functions

- [CertificateRequest](#) ()
- [CertificateRequest](#) (const **QString** &fileName)
- [CertificateRequest](#) (const [CertificateOptions](#) &opts, const [PrivateKey](#) &key, const **QString** &provider=**QString**())
- [CertificateRequest](#) (const [CertificateRequest](#) &from)
- [CertificateRequest](#) & operator= (const [CertificateRequest](#) &from)
- bool [isNull](#) () const
- [CertificateRequestFormat](#) [format](#) () const
- **CertificateInfo** [subjectInfo](#) () const
- [CertificateInfoOrdered](#) [subjectInfoOrdered](#) () const
- **Constraints** [constraints](#) () const
- **QStringList** [policies](#) () const
- [PublicKey](#) [subjectPublicKey](#) () const
- bool [isCA](#) () const
- int [pathLimit](#) () const
- **QString** [challenge](#) () const
- [SignatureAlgorithm](#) [signatureAlgorithm](#) () const
- bool [operator==](#) (const [CertificateRequest](#) &csr) const
- bool [operator!=](#) (const [CertificateRequest](#) &other) const
- **QByteArray** [toDER](#) () const
- **QString** [toPEM](#) () const
- bool [toPEMFile](#) (const **QString** &fileName) const
- **QString** [toString](#) () const
- void [change](#) (CSRContext *c)

Static Public Member Functions

- static bool [canUseFormat](#) ([CertificateRequestFormat](#) f, const [QString](#) &provider=[QString](#)())
- static [CertificateRequest fromDER](#) (const [QByteArray](#) &a, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [CertificateRequest fromPEM](#) (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [CertificateRequest fromPEMFile](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [CertificateRequest fromString](#) (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

Friends

- class [Private](#)

10.15.1 Detailed Description

Certificate Request

A [CertificateRequest](#) is a unsigned request for a [Certificate](#)

10.15.2 Constructor & Destructor Documentation

10.15.2.1 QCA::CertificateRequest::CertificateRequest ()

Create an empty certificate request.

10.15.2.2 QCA::CertificateRequest::CertificateRequest (const [QString](#) &fileName)

Create a certificate request based on the contents of a file.

Parameters:

fileName the file (and path, if necessary) containing a PEM encoded certificate request

10.15.2.3 QCA::CertificateRequest::CertificateRequest (const [CertificateOptions](#) &opts, const [PrivateKey](#) &key, const [QString](#) &provider = [QString](#) ())

Create a certificate request based on specified options.

Parameters:

opts the options to use in the certificate request

key the private key that matches the certificate being requested

provider the provider to use, if a specific provider is required

10.15.2.4 QCA::CertificateRequest::CertificateRequest (const [CertificateRequest](#) &from)

Standard copy constructor.

10.15.3 Member Function Documentation

10.15.3.1 [CertificateRequest](#)& QCA::CertificateRequest::operator= (const [CertificateRequest](#) & *from*)

Standard assignment operator.

10.15.3.2 bool QCA::CertificateRequest::isNull () const

test if the certificate request is empty

Returns:

true if the certificate request is empty, otherwise false

10.15.3.3 static bool QCA::CertificateRequest::canUseFormat ([CertificateRequestFormat](#) *f*, const QString & *provider* = QString ()) [static]

Test if the certificate request can use a specified format.

Parameters:

f the format to test for

provider the provider to use, if a specific provider is required

Returns:

true if the certificate request can use the specified format

10.15.3.4 [CertificateRequestFormat](#) QCA::CertificateRequest::format () const

the format that this [Certificate](#) request is in

10.15.3.5 [CertificateInfo](#) QCA::CertificateRequest::subjectInfo () const

Information on the subject of the certificate being requested.

Note:

this only applies to PKCS#10 format certificate requests

See also:

[subjectInfoOrdered](#) for a version that maintains order in the subject information.

10.15.3.6 [CertificateInfoOrdered](#) QCA::CertificateRequest::subjectInfoOrdered () const

Information on the subject of the certificate being requested, as an ordered list ([QList](#) of [CertificateInfoPair](#)).

Note:

this only applies to PKCS#10 format certificate requests

See also:

[subjectInfo](#) for a version that does not maintain order, but allows access based on a multimap.
[CertificateInfoPair](#) for the elements in the list

10.15.3.7 Constraints QCA::CertificateRequest::constraints () const

The constraints that apply to this certificate request.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.8 QStringList QCA::CertificateRequest::policies () const

The policies that apply to this certificate request.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.9 PublicKey QCA::CertificateRequest::subjectPublicKey () const

The public key belonging to the issuer.

10.15.3.10 bool QCA::CertificateRequest::isCA () const

Test if this [Certificate](#) Request is for a [Certificate](#) Authority certificate.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.11 int QCA::CertificateRequest::pathLimit () const

The path limit for the certificate in this [Certificate](#) Request.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.12 QString QCA::CertificateRequest::challenge () const

The challenge associated with this certificate request.

10.15.3.13 SignatureAlgorithm QCA::CertificateRequest::signatureAlgorithm () const

The algorithm used to make the signature on this certificate request.

10.15.3.14 `bool QCA::CertificateRequest::operator==(const CertificateRequest & csr) const`

Test for equality of two certificate requests.

Returns:

true if the two certificate requests are the same

10.15.3.15 `bool QCA::CertificateRequest::operator!=(const CertificateRequest & other) const`
[inline]

Inequality operator.

10.15.3.16 `QByteArray QCA::CertificateRequest::toDER () const`

Export the [Certificate](#) Request into a DER format.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.17 `QString QCA::CertificateRequest::toPEM () const`

Export the [Certificate](#) Request into a PEM format.

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.18 `bool QCA::CertificateRequest::toPEMFile (const QString & fileName) const`

Export the [Certificate](#) into PEM format in a file.

Parameters:

fileName the name of the file to use

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.19 `static CertificateRequest QCA::CertificateRequest::fromDER (const QByteArray & a,
ConvertResult * result = 0, const QString & provider = QString())` [static]

Import the certificate request from DER.

Parameters:

a the array containing the certificate request in DER format

result a pointer to a ConvertResult, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CertificateRequest](#) corresponding to the certificate request in the provided array

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.20 static [CertificateRequest](#) QCA::CertificateRequest::fromPEM (const QString & *s*,
[ConvertResult](#) * *result* = 0, const QString & *provider* = QString ()) [static]

Import the certificate request from PEM format.

Parameters:

s the string containing the certificate request in PEM format

result a pointer to a ConvertResult, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CertificateRequest](#) corresponding to the certificate request in the provided string

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.21 static [CertificateRequest](#) QCA::CertificateRequest::fromPEMFile (const QString
& *fileName*, [ConvertResult](#) * *result* = 0, const QString & *provider* = QString ())
[static]

Import the certificate request from a file.

Parameters:

fileName the name (and path, if required) of the file containing the certificate request in PEM format

result a pointer to a ConvertResult, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CertificateRequest](#) corresponding to the certificate request in the provided string

Note:

this only applies to PKCS#10 format certificate requests

10.15.3.22 QString QCA::CertificateRequest::toString () const

Export the [CertificateRequest](#) to a string.

Returns:

the string corresponding to the certificate request

Note:

this only applies to SPKAC format certificate requests

10.15.3.23 **static [CertificateRequest](#) QCA::CertificateRequest::fromString (const QString & *s*, [ConvertResult](#) * *result* = 0, const QString & *provider* = QString())** [static]

Import the [CertificateRequest](#) from a string.

Parameters:

- s* the string containing to the certificate request
- result* a pointer to a [ConvertResult](#), which if not-null will be set to the conversion status
- provider* the provider to use, if a specific provider is required

Returns:

the [CertificateRequest](#) corresponding to the certificate request in the provided string

Note:

this only applies to SPKAC format certificate requests

10.15.3.24 **void QCA::CertificateRequest::change (CSRContext * *c*)**

For internal use only.

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

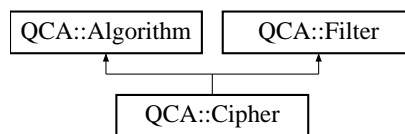
- [qca_cert.h](#)

10.16 QCA::Cipher Class Reference

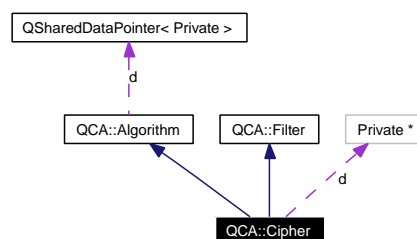
General class for cipher (encryption / decryption) algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Cipher:



Collaboration diagram for QCA::Cipher:



Public Types

- enum [Mode](#) { [CBC](#), [CFB](#), [ECB](#), [OFB](#) }
- enum [Padding](#) { [DefaultPadding](#), [NoPadding](#), [PKCS7](#) }

Public Member Functions

- [Cipher](#) (const [QString](#) &type, [Mode](#) mode, [Padding](#) pad=DefaultPadding, [Direction](#) dir=Encode, const [SymmetricKey](#) &key=[SymmetricKey](#)(), const [InitializationVector](#) &iv=[InitializationVector](#)(), const [QString](#) &provider=[QString](#)())
- [Cipher](#) (const [Cipher](#) &from)
- [Cipher](#) & operator= (const [Cipher](#) &from)
- [QString](#) type () const
- [Mode](#) mode () const
- [Padding](#) padding () const
- [Direction](#) direction () const
- [KeyLength](#) keyLength () const
- bool [validKeyLength](#) (int n) const
- int [blockSize](#) () const
- virtual void [clear](#) ()
- virtual [MemoryRegion](#) [update](#) (const [MemoryRegion](#) &a)
- virtual [MemoryRegion](#) [final](#) ()
- virtual bool [ok](#) () const
- void [setup](#) ([Direction](#) dir, const [SymmetricKey](#) &key, const [InitializationVector](#) &iv=[InitializationVector](#)())

Static Public Member Functions

- static **QString** [withAlgorithms](#) (const **QString** &cipherType, [Mode](#) modeType, [Padding](#) paddingType)

10.16.1 Detailed Description

General class for cipher (encryption / decryption) algorithms.

[Cipher](#) is the class for the various algorithms that perform low level encryption and decryption within QCA.

AES128, AES192 and AES256 are recommended for new applications.

Standard names for ciphers are:

- Blowfish - "blowfish"
- TripleDES - "tripledes"
- DES - "des"
- AES128 - "aes128"
- AES192 - "aes192"
- AES256 - "aes256"

Examples:

[aes-cmac.cpp](#), and [ciphertest.cpp](#).

10.16.2 Member Enumeration Documentation

10.16.2.1 enum [QCA::Cipher::Mode](#)

Mode settings for cipher algorithms.

Enumerator:

- CBC*** operate in Cipher Block Chaining mode
- CFB*** operate in Cipher FeedBack mode
- ECB*** operate in Electronic Code Book mode
- OFB*** operate in Output FeedBack Mode

10.16.2.2 enum [QCA::Cipher::Padding](#)

Padding variations for cipher algorithms.

Enumerator:

- DefaultPadding*** Default for cipher-mode.
- NoPadding*** Do not use padding.
- PKCS7*** Pad using the scheme in PKCS#7.

10.16.3 Constructor & Destructor Documentation

10.16.3.1 `QCA::Cipher::Cipher (const QString & type, Mode mode, Padding pad = DefaultPadding, Direction dir = Encode, const SymmetricKey & key = SymmetricKey(), const InitializationVector & iv = InitializationVector(), const QString & provider = QString())`

Standard constructor.

Parameters:

- type* the name of the cipher specialisation to use (e.g. "aes128")
- mode* the operating Mode to use (e.g. [QCA::Cipher::CBC](#))
- pad* the type of Padding to use
- dir* the Direction that this [Cipher](#) should use (Encode for encryption, Decode for decryption)
- key* the [SymmetricKey](#) array that is the key
- iv* the [InitializationVector](#) to use (not used for ECB mode)
- provider* the name of the [Provider](#) to use

Note:

Padding only applies to CBC and ECB modes. CFB and OFB ciphertext is always the length of the plaintext.

10.16.3.2 `QCA::Cipher::Cipher (const Cipher & from)`

Standard copy constructor.

10.16.4 Member Function Documentation

10.16.4.1 `Cipher& QCA::Cipher::operator= (const Cipher & from)`

Assignment operator.

Parameters:

- from* the [Cipher](#) to copy state from

10.16.4.2 `QString QCA::Cipher::type () const`

Return the cipher type.

Reimplemented from [QCA::Algorithm](#).

10.16.4.3 `Mode QCA::Cipher::mode () const`

Return the cipher mode.

10.16.4.4 `Padding QCA::Cipher::padding () const`

Return the cipher padding type.

10.16.4.5 [Direction](#) QCA::Cipher::direction () const

Return the cipher direction.

10.16.4.6 [KeyLength](#) QCA::Cipher::keyLength () const

Return acceptable key lengths.

10.16.4.7 bool QCA::Cipher::validKeyLength (int *n*) const

Test if a key length is valid for the cipher algorithm.

Parameters:

n the key length in bytes

Returns:

true if the key would be valid for the current algorithm

10.16.4.8 int QCA::Cipher::blockSize () const

return the block size for the cipher object

10.16.4.9 virtual void QCA::Cipher::clear () [virtual]

reset the cipher object, to allow re-use

Implements [QCA::Filter](#).

**10.16.4.10 virtual [MemoryRegion](#) QCA::Cipher::update (const [MemoryRegion](#) & *a*)
[virtual]**

pass in a byte array of data, which will be encrypted or decrypted (according to the Direction that was set in the constructor or in [setup\(\)](#)) and returned.

Parameters:

a the array of data to encrypt / decrypt

Implements [QCA::Filter](#).

Examples:

[ciphertest.cpp](#).

10.16.4.11 virtual [MemoryRegion](#) QCA::Cipher::final () [virtual]

complete the block of data, padding as required, and returning the completed block

Implements [QCA::Filter](#).

10.16.4.12 `virtual bool QCA::Cipher::ok () const` [virtual]

Test if an [update\(\)](#) or [final\(\)](#) call succeeded.

Returns:

true if the previous call succeeded

Implements [QCA::Filter](#).

Examples:

[ciphertest.cpp](#).

10.16.4.13 `void QCA::Cipher::setup (Direction dir, const SymmetricKey & key, const InitializationVector & iv = InitializationVector ())`

Reset / reconfigure the [Cipher](#).

You can use this to re-use an existing [Cipher](#), rather than creating a new object with a slightly different configuration.

Parameters:

dir the Direction that this [Cipher](#) should use (Encode for encryption, Decode for decryption)

key the [SymmetricKey](#) array that is the key

iv the [InitializationVector](#) to use

10.16.4.14 `static QString QCA::Cipher::withAlgorithms (const QString & cipherType, Mode modeType, Padding paddingType)` [static]

Construct a [Cipher](#) type string.

Parameters:

cipherType the name of the algorithm (eg AES128, DES)

modeType the mode to operate the cipher in (eg QCA::CBC, QCA::CFB)

paddingType the padding required (eg QCA::NoPadding, QCA::PKCS7)

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

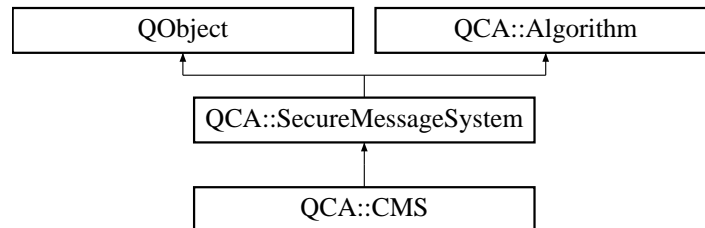
- [qca_basic.h](#)

10.17 QCA::CMS Class Reference

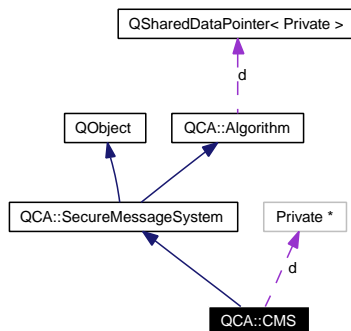
Cryptographic Message Syntax messaging system.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::CMS:



Collaboration diagram for QCA::CMS:



Public Member Functions

- `CMS` (`QObject *parent=0`, `const QString &provider=QString()`)
- `CertificateCollection trustedCertificates` () const
- `CertificateCollection untrustedCertificates` () const
- `SecureMessageKeyList privateKeys` () const
- void `setTrustedCertificates` (const `CertificateCollection` &trusted)
- void `setUntrustedCertificates` (const `CertificateCollection` &untrusted)
- void `setPrivateKeys` (const `SecureMessageKeyList` &keys)

10.17.1 Detailed Description

Cryptographic Message Syntax messaging system.

Cryptographic Message Syntax (CMS) "is used to digitally sign, digest, authenticate, or encrypt arbitrary message content. The CMS describes an encapsulation syntax for data protection. It supports digital signatures and encryption. The syntax allows multiple encapsulations; one encapsulation envelope can be nested inside another. Likewise, one party can digitally sign some previously encapsulated data. It also allows arbitrary attributes, such as signing time, to be signed along with the message content, and provides for other attributes such as countersignatures to be associated with a signature." (from [RFC3852](#) "Cryptographic Message Syntax")

See also:

[SecureMessage](#)
[SecureMessageKey](#)

Examples:

[cmsexample.cpp](#), and [publickeyexample.cpp](#).

10.17.2 Constructor & Destructor Documentation

10.17.2.1 QCA::CMS::CMS (QObject * *parent* = 0, const QString & *provider* = QString ()) [explicit]

Standard constructor.

Parameters:

parent the parent object for this object

provider the provider to use, if a specific provider is required

10.17.3 Member Function Documentation

10.17.3.1 [CertificateCollection](#) QCA::CMS::trustedCertificates () const

Return the trusted certificates set for this object.

10.17.3.2 [CertificateCollection](#) QCA::CMS::untrustedCertificates () const

Return the untrusted certificates set for this object.

10.17.3.3 [SecureMessageKeyList](#) QCA::CMS::privateKeys () const

Return the private keys set for this object.

10.17.3.4 void QCA::CMS::setTrustedCertificates (const [CertificateCollection](#) & *trusted*)

Set the trusted certificates to use for the messages built using this [CMS](#) object.

Parameters:

trusted the collection of trusted certificates to use

10.17.3.5 void QCA::CMS::setUntrustedCertificates (const [CertificateCollection](#) & *untrusted*)

Set the untrusted certificates to use for the messages built using this [CMS](#) object.

This function is useful when verifying messages that don't contain the certificates (or intermediate signers) within the [CMS](#) blob. In order to verify such messages, you'll have to pass the possible signer certs with this function.

Parameters:

untrusted the collection of untrusted certificates to use

10.17.3.6 void QCA::CMS::setPrivateKeys (const SecureMessageKeyList & *keys*)

Set the private keys to use for the messages built using this [CMS](#) object.

Keys are required for decrypting and signing (not for encrypting or verifying).

Parameters:

keys the collection of keys to use

Examples:

[cmsexample.cpp](#).

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

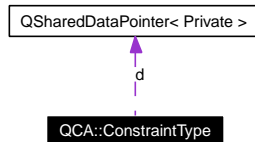
- [qca_securemessage.h](#)

10.18 QCA::ConstraintType Class Reference

[Certificate](#) constraint.

```
#include <qca_cert.h>
```

Collaboration diagram for QCA::ConstraintType:



Public Types

- enum [Section](#) { [KeyUsage](#), [ExtendedKeyUsage](#) }

Public Member Functions

- [ConstraintType](#) ()
- [ConstraintType](#) ([ConstraintTypeKnown](#) known)
- [ConstraintType](#) (const **QString** &id, [Section](#) section)
- [ConstraintType](#) (const [ConstraintType](#) &from)
- [ConstraintType](#) & operator= (const [ConstraintType](#) &from)
- [Section](#) section () const
- [ConstraintTypeKnown](#) known () const
- **QString** id () const
- bool operator< (const [ConstraintType](#) &other) const
- bool operator== (const [ConstraintType](#) &other) const
- bool operator!= (const [ConstraintType](#) &other) const

10.18.1 Detailed Description

[Certificate](#) constraint.

X.509 certificates can be constrained in their application - that is, some certificates can only be used for certain purposes. This class is used to identify an approved purpose for a certificate.

Note:

It is common for a certificate to have more than one purpose.

10.18.2 Member Enumeration Documentation

10.18.2.1 enum [QCA::ConstraintType::Section](#)

Section of the certificate that the constraint belongs in.

Enumerator:

KeyUsage Stored in the key usage section.

ExtendedKeyUsage Stored in the extended key usage section.

10.18.3 Constructor & Destructor Documentation

10.18.3.1 QCA::ConstraintType::ConstraintType ()

Standard constructor.

10.18.3.2 QCA::ConstraintType::ConstraintType ([ConstraintTypeKnown](#) *known*)

Construct a new constraint.

The section will be derived by *known*.

Parameters:

known the type as part of the ConstraintTypeKnown enumerator

10.18.3.3 QCA::ConstraintType::ConstraintType (const QString & *id*, [Section](#) *section*)

Construct a new constraint.

Parameters:

id the type as an identifier string (OID or internal)

section the section this type belongs in

See also:

[id](#)

10.18.3.4 QCA::ConstraintType::ConstraintType (const [ConstraintType](#) & *from*)

Standard copy constructor.

10.18.4 Member Function Documentation

10.18.4.1 [ConstraintType&](#) QCA::ConstraintType::operator= (const [ConstraintType](#) & *from*)

Standard assignment operator.

10.18.4.2 [Section](#) QCA::ConstraintType::section () const

The section the constraint is part of.

10.18.4.3 [ConstraintTypeKnown](#) QCA::ConstraintType::known () const

The type as part of the ConstraintTypeKnown enumerator.

This function may return a value that does not exist in the enumerator. In that case, you may use [id\(\)](#) to determine the type.

10.18.4.4 QString QCA::ConstraintType::id () const

The type as an identifier string.

For types that have OIDs, this function returns an OID in string form. For types that do not have OIDs, this function returns an internal identifier string whose first character is not a digit (this allows you to tell the difference between an OID and an internal identifier).

It is hereby stated that the KeyUsage bit fields shall use the internal identifier format "KeyUsage.[rfc field name]". For example, the keyEncipherment field would have the identifier "KeyUsage.keyEncipherment".

Applications should not store, use, or compare against internal identifiers unless the identifiers are explicitly documented (e.g. KeyUsage).

10.18.4.5 bool QCA::ConstraintType::operator< (const ConstraintType & other) const

Comparison operator.

10.18.4.6 bool QCA::ConstraintType::operator== (const ConstraintType & other) const

Comparison operator.

10.18.4.7 bool QCA::ConstraintType::operator!= (const ConstraintType & other) const [inline]

Inequality operator.

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

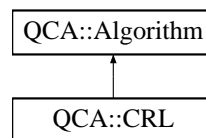
- [qca_cert.h](#)

10.19 QCA::CRL Class Reference

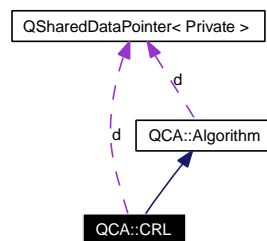
Certificate Revocation List

```
#include <QtCrypto>
```

Inheritance diagram for QCA::CRL::



Collaboration diagram for QCA::CRL:



Public Member Functions

- [CRL](#) (const [CRL](#) &from)
- [CRL](#) & [operator=](#) (const [CRL](#) &from)
- bool [isNull](#) () const
- [CertificateInfo](#) [issuerInfo](#) () const
- [CertificateInfoOrdered](#) [issuerInfoOrdered](#) () const
- int [number](#) () const
- [QDateTime](#) [thisUpdate](#) () const
- [QDateTime](#) [nextUpdate](#) () const
- [QList](#)< [CRLEntry](#) > [revoked](#) () const
- [SignatureAlgorithm](#) [signatureAlgorithm](#) () const
- [QByteArray](#) [issuerKeyId](#) () const
- bool [operator==](#) (const [CRL](#) &a) const
- bool [operator!=](#) (const [CRL](#) &other) const
- [QByteArray](#) [toDER](#) () const
- [QString](#) [toPEM](#) () const
- bool [toPEMFile](#) (const [QString](#) &fileName) const
- void [change](#) (CRLContext *c)

Static Public Member Functions

- static [CRL](#) [fromDER](#) (const [QByteArray](#) &a, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

- static [CRL fromPEM](#) (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [CRL fromPEMFile](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

Friends

- class [Private](#)

10.19.1 Detailed Description

Certificate Revocation List

A CRL is a list of certificates that are special in some way. The normal reason for including a certificate on a CRL is that the certificate should no longer be used. For example, if a key is compromised, then the associated certificate may no longer provides appropriate security. There are other reasons why a certificate may be placed on a CRL, as shown in the [CRLEntry::Reason](#) enumeration.

See also:

[CertificateCollection](#) for a way to handle Certificates and CRLs as a single entity.
[CRLEntry](#) for the CRL segment representing a single [Certificate](#).

10.19.2 Constructor & Destructor Documentation

10.19.2.1 QCA::CRL::CRL (const [CRL](#) &from)

Standard copy constructor.

10.19.3 Member Function Documentation

10.19.3.1 [CRL&](#) QCA::CRL::operator= (const [CRL](#) &from)

Standard assignment operator.

10.19.3.2 bool QCA::CRL::isNull () const

Test if the [CRL](#) is empty.

Returns:

true if the [CRL](#) is entry, otherwise return false

10.19.3.3 CertificateInfo QCA::CRL::issuerInfo () const

Information on the issuer of the [CRL](#) as a [QMultiMap](#).

See also:

[issuerInfoOrdered](#) for a version that maintains the order of information fields as per the underlying [CRL](#).

10.19.3.4 [CertificateInfoOrdered](#) QCA::CRL::issuerInfoOrdered () const

Information on the issuer of the [CRL](#) as an ordered list ([QList](#) of [CertificateInfoPair](#)).

See also:

[issuerInfo](#) for a version that allows lookup based on a multimap.
[CertificateInfoPair](#) for the elements in the list

10.19.3.5 `int` QCA::CRL::number () const

The [CRL](#) serial number.

Note that serial numbers are a [CRL](#) extension, and not all certificates have one.

Returns:

the [CRL](#) serial number, or -1 if there is no serial number

10.19.3.6 `QDateTime` QCA::CRL::thisUpdate () const

the time that this [CRL](#) became (or becomes) valid

10.19.3.7 `QDateTime` QCA::CRL::nextUpdate () const

the time that this [CRL](#) will be obsoleted

you should obtain an updated [CRL](#) at this time

10.19.3.8 `QList<CRLEntry>` QCA::CRL::revoked () const

a list of the revoked certificates in this [CRL](#)

10.19.3.9 [SignatureAlgorithm](#) QCA::CRL::signatureAlgorithm () const

The signature algorithm used for the signature on this [CRL](#).

10.19.3.10 `QByteArray` QCA::CRL::issuerKeyId () const

The key identification of the [CRL](#) issuer.

10.19.3.11 `bool` QCA::CRL::operator== (const [CRL](#) & *a*) const

Test for equality of two Certificate Revocation Lists.

Returns:

true if the two CRLs are the same

10.19.3.12 `bool QCA::CRL::operator!=(const CRL & other) const` `[inline]`

Inequality operator.

10.19.3.13 `QByteArray QCA::CRL::toDER () const`

Export the Certificate Revocation List ([CRL](#)) in DER format.

Returns:

an array containing the [CRL](#) in DER format

10.19.3.14 `QString QCA::CRL::toPEM () const`

Export the Certificate Revocation List ([CRL](#)) in PEM format.

Returns:

a string containing the [CRL](#) in PEM format

10.19.3.15 `bool QCA::CRL::toPEMFile (const QString & fileName) const`

Export the Certificate Revocation List ([CRL](#)) into PEM format in a file.

Parameters:

fileName the name of the file to use

10.19.3.16 `static CRL QCA::CRL::fromDER (const QByteArray & a, ConvertResult * result = 0, const QString & provider = QString ())` `[static]`

Import a DER encoded Certificate Revocation List ([CRL](#)).

Parameters:

a the array containing the [CRL](#) in DER format

result a pointer to a [ConvertResult](#), which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CRL](#) corresponding to the contents of the array

10.19.3.17 `static CRL QCA::CRL::fromPEM (const QString & s, ConvertResult * result = 0, const QString & provider = QString ())` `[static]`

Import a PEM encoded Certificate Revocation List ([CRL](#)).

Parameters:

s the string containing the [CRL](#) in PEM format

result a pointer to a `ConvertResult`, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CRL](#) corresponding to the contents of the string

10.19.3.18 static [CRL](#) `QCA::CRL::fromPEMFile (const QString & fileName, ConvertResult * result = 0, const QString & provider = QString())` [static]

Import a PEM encoded Certificate Revocation List ([CRL](#)) from a file.

Parameters:

fileName the name (and path, if required) of the file containing the certificate in PEM format

result a pointer to a `ConvertResult`, which if not-null will be set to the conversion status

provider the provider to use, if a specific provider is required

Returns:

the [CRL](#) in the file

10.19.3.19 void `QCA::CRL::change (CRLContext * c)`

For internal use only.

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

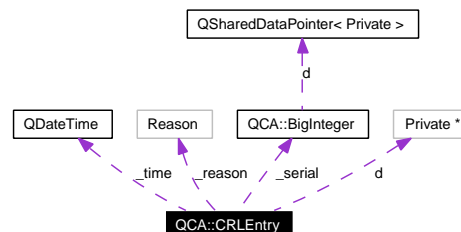
- [qca_cert.h](#)

10.20 QCA::CRLEntry Class Reference

Part of a [CRL](#) representing a single certificate.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::CRLEntry:



Public Types

- enum [Reason](#) {
[Unspecified](#), [KeyCompromise](#), [CACompromise](#), [AffiliationChanged](#),
[Superseded](#), [CessationOfOperation](#), [CertificateHold](#), [RemoveFromCRL](#),
[PrivilegeWithdrawn](#), [AACompromise](#) }

Public Member Functions

- [CRLEntry](#) ()
- [CRLEntry](#) (const [Certificate](#) &c, [Reason](#) r=[Unspecified](#))
- [CRLEntry](#) (const [BigInteger](#) serial, const [QDateTime](#) &time, [Reason](#) r=[Unspecified](#))
- [CRLEntry](#) (const [CRLEntry](#) &from)
- [CRLEntry](#) & [operator=](#) (const [CRLEntry](#) &from)
- [BigInteger](#) [serialNumber](#) () const
- [QDateTime](#) [time](#) () const
- bool [isNull](#) () const
- [Reason](#) [reason](#) () const
- bool [operator<](#) (const [CRLEntry](#) &a) const
- bool [operator==](#) (const [CRLEntry](#) &a) const
- bool [operator!=](#) (const [CRLEntry](#) &other) const

10.20.1 Detailed Description

Part of a [CRL](#) representing a single certificate.

10.20.2 Member Enumeration Documentation

10.20.2.1 enum [QCA::CRLEntry::Reason](#)

The reason why the certificate has been revoked.

Enumerator:

- Unspecified* reason is unknown
- KeyCompromise* private key has been compromised
- CACompromise* certificate authority has been compromised
- Superseded* certificate has been superseded
- CertificateHold* certificate is on hold
- RemoveFromCRL* certificate was previously in a [CRL](#), but is now valid
- AACompromise* attribute authority has been compromised

10.20.3 Constructor & Destructor Documentation

10.20.3.1 QCA::CRLEntry::CRLEntry ()

create an empty [CRL](#) entry

10.20.3.2 QCA::CRLEntry::CRLEntry (const [Certificate](#) & *c*, [Reason](#) *r* = Unspecified) [explicit]

create a [CRL](#) entry

Parameters:

- c* the certificate to revoke
- r* the reason that the certificate is being revoked

10.20.3.3 QCA::CRLEntry::CRLEntry (const [BigInteger](#) *serial*, const QDateTime & *time*, [Reason](#) *r* = Unspecified)

create a [CRL](#) entry

Parameters:

- serial* the serial number of the [Certificate](#) being revoked
- time* the time the [Certificate](#) was revoked (or will be revoked)
- r* the reason that the certificate is being revoked

10.20.3.4 QCA::CRLEntry::CRLEntry (const [CRLEntry](#) & *from*)

Copy constructor.

Parameters:

- from* the [CRLEntry](#) to copy from

10.20.4 Member Function Documentation

10.20.4.1 [CRLEntry](#) & QCA::CRLEntry::operator= (const [CRLEntry](#) & *from*)

Standard assignment operator.

Parameters:

from the [CRLEntry](#) to copy from

10.20.4.2 [BigInteger](#) QCA::CRLEntry::serialNumber () const

The serial number of the certificate that is the subject of this [CRL](#) entry.

10.20.4.3 [QDateTime](#) QCA::CRLEntry::time () const

The time this [CRL](#) entry was created.

10.20.4.4 [bool](#) QCA::CRLEntry::isNull () const

Test if this [CRL](#) entry is empty.

10.20.4.5 [Reason](#) QCA::CRLEntry::reason () const

The reason that this [CRL](#) entry was created.

Alternatively, you might like to think of this as the reason that the subject certificate has been revoked

10.20.4.6 [bool](#) QCA::CRLEntry::operator< (const [CRLEntry](#) & *a*) const

Test if one [CRL](#) entry is "less than" another.

[CRL](#) entries are compared based on their serial number

10.20.4.7 [bool](#) QCA::CRLEntry::operator== (const [CRLEntry](#) & *a*) const

Test for equality of two [CRL](#) Entries.

Returns:

true if the two certificates are the same

10.20.4.8 [bool](#) QCA::CRLEntry::operator!= (const [CRLEntry](#) & *other*) const [inline]

Inequality operator.

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

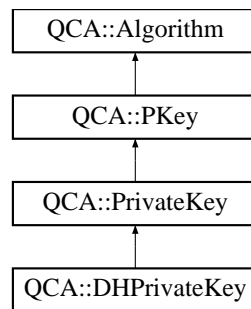
- [qca_cert.h](#)

10.21 QCA::DHPrivateKey Class Reference

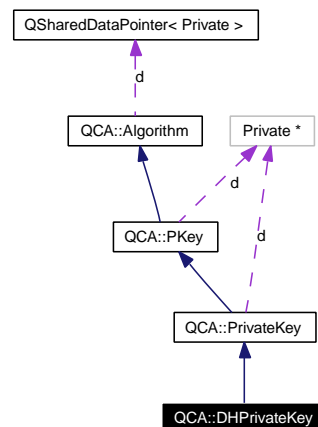
Diffie-Hellman Private Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::DHPrivateKey::



Collaboration diagram for QCA::DHPrivateKey:



Public Member Functions

- `DHPrivateKey ()`
- `DHPrivateKey (const DLGroup &domain, const BigInteger &y, const BigInteger &x, const QString &provider=QString())`
- `DLGroup domain () const`
- `BigInteger y () const`
- `BigInteger x () const`

10.21.1 Detailed Description

Diffie-Hellman Private Key.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 QCA::DHPrivateKey::DHPrivateKey ()

Create an empty Diffie-Hellman private key.

10.21.2.2 QCA::DHPrivateKey::DHPrivateKey (const [DLGroup](#) & *domain*, const [BigInteger](#) & *y*, const [BigInteger](#) & *x*, const QString & *provider* = QString ())

Create a Diffie-Hellman private key.

Parameters:

domain the discrete logarithm group to use

y the public random value

x the private random value

provider the provider to use, if a particular provider is required

10.21.3 Member Function Documentation

10.21.3.1 [DLGroup](#) QCA::DHPrivateKey::domain () const

The discrete logarithm group that is being used.

10.21.3.2 [BigInteger](#) QCA::DHPrivateKey::y () const

The public random value associated with this key.

10.21.3.3 [BigInteger](#) QCA::DHPrivateKey::x () const

The private random value associated with this key.

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

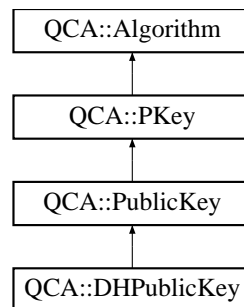
- [qca_publickey.h](#)

10.22 QCA::DHPublicKey Class Reference

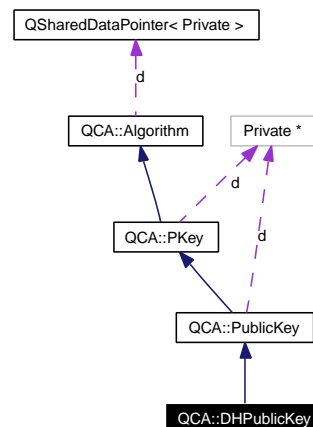
Diffie-Hellman Public Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::DHPublicKey::



Collaboration diagram for QCA::DHPublicKey:



Public Member Functions

- `DHPublicKey` ()
- `DHPublicKey` (const `DLGroup` &domain, const `BigInteger` &y, const `QString` &provider=`QString`())
- `DHPublicKey` (const `DHPrivateKey` &k)
- `DLGroup domain` () const
- `BigInteger y` () const

10.22.1 Detailed Description

Diffie-Hellman Public Key.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 QCA::DHPrivateKey::DHPrivateKey ()

Create an empty Diffie-Hellman public key.

10.22.2.2 QCA::DHPrivateKey::DHPrivateKey (const [DLGroup](#) & *domain*, const [BigInteger](#) & *y*, const [QString](#) & *provider* = [QString](#) ())

Create a Diffie-Hellman public key.

Parameters:

domain the discrete logarithm group to use

y the public random value

provider the provider to use, if a specific provider is required

10.22.2.3 QCA::DHPrivateKey::DHPrivateKey (const [DHPrivateKey](#) & *k*)

Create a Diffie-Hellman public key from a specified private key.

Parameters:

k the Diffie-Hellman private key to use as the source

10.22.3 Member Function Documentation

10.22.3.1 [DLGroup](#) QCA::DHPrivateKey::domain () const

The discrete logarithm group that is being used.

10.22.3.2 [BigInteger](#) QCA::DHPrivateKey::y () const

The public random value associated with this key.

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

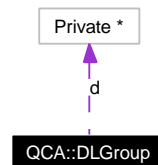
- [qca_publickey.h](#)

10.23 QCA::DLGroup Class Reference

A discrete logarithm group.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::DLGroup:



Public Member Functions

- `DLGroup` (const `BigInteger` &p, const `BigInteger` &q, const `BigInteger` &g)
- `DLGroup` (const `BigInteger` &p, const `BigInteger` &g)
- `DLGroup` (const `DLGroup` &from)
- `DLGroup` & `operator=` (const `DLGroup` &from)
- `bool isNull` () const
- `BigInteger p` () const
- `BigInteger q` () const
- `BigInteger g` () const

Static Public Member Functions

- static `QList< DLGroupSet > supportedGroupSets` (const `QString` &provider=`QString()`)

10.23.1 Detailed Description

A discrete logarithm group.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 QCA::DLGroup::DLGroup (const `BigInteger` & p, const `BigInteger` & q, const `BigInteger` & g)

Construct a discrete logarithm group from raw parameters.

Parameters:

p
q
g

10.23.2.2 QCA::DLGroup::DLGroup (const BigInteger & *p*, const BigInteger & *g*)

Construct a discrete logarithm group from raw parameters.

Parameters:

p

g

10.23.2.3 QCA::DLGroup::DLGroup (const DLGroup & *from*)

Standard copy constructor.

10.23.3 Member Function Documentation

10.23.3.1 DLGroup& QCA::DLGroup::operator= (const DLGroup & *from*)

Standard assignment operator.

Parameters:

from the DLGroup to copy from

10.23.3.2 static QList<DLGroupSet> QCA::DLGroup::supportedGroupSets (const QString & *provider* = QString ()) [static]

Provide a list of the supported group sets.

Parameters:

provider the provider to report which group sets are available. If not specified, all providers will be checked

10.23.3.3 bool QCA::DLGroup::isNull () const

Test if the group is empty.

10.23.3.4 BigInteger QCA::DLGroup::p () const

Provide the p component of the group.

10.23.3.5 BigInteger QCA::DLGroup::q () const

Provide the q component of the group.

10.23.3.6 [BigInteger](#) QCA::DLGroup::g () const

Provide the g component of the group.

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

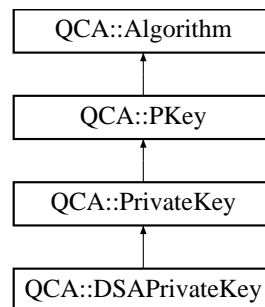
- [qca_publickey.h](#)

10.24 QCA::DSAPrivateKey Class Reference

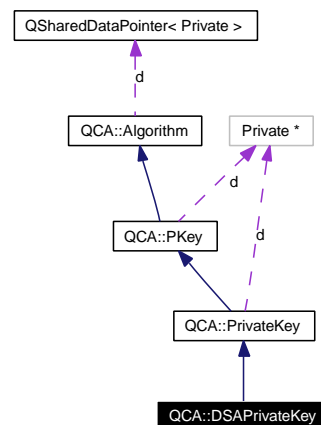
Digital Signature Algorithm Private Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::DSAPrivateKey::



Collaboration diagram for QCA::DSAPrivateKey:



Public Member Functions

- [DSAPrivateKey](#) ()
- [DSAPrivateKey](#) (const [DLGroup](#) &domain, const [BigInteger](#) &y, const [BigInteger](#) &x, const [QString](#) &provider=[QString](#)())
- [DLGroup domain](#) () const
- [BigInteger y](#) () const
- [BigInteger x](#) () const

10.24.1 Detailed Description

Digital Signature Algorithm Private Key.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 QCA::DSAPrivateKey::DSAPrivateKey ()

Create an empty DSA private key.

10.24.2.2 QCA::DSAPrivateKey::DSAPrivateKey (const [DLGroup](#) & *domain*, const [BigInteger](#) & *y*, const [BigInteger](#) & *x*, const QString & *provider* = QString ())

Create a DSA public key.

Parameters:

domain the discrete logarithm group to use

y the public random value

x the private random value

provider the provider to use, if a specific provider is required

10.24.3 Member Function Documentation

10.24.3.1 [DLGroup](#) QCA::DSAPrivateKey::domain () const

The discrete logarithm group that is being used.

10.24.3.2 [BigInteger](#) QCA::DSAPrivateKey::y () const

the public random value

10.24.3.3 [BigInteger](#) QCA::DSAPrivateKey::x () const

the private random value

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

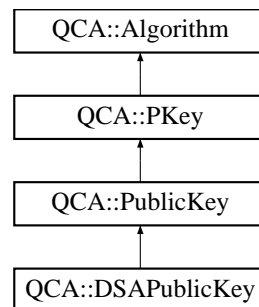
- [qca_publickey.h](#)

10.25 QCA::DSAPublicKey Class Reference

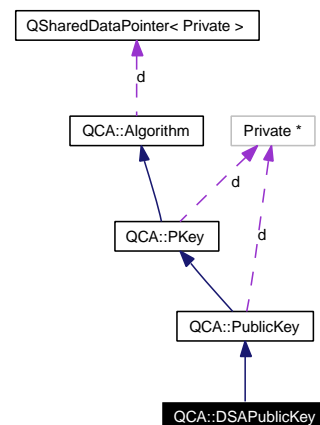
Digital Signature Algorithm Public Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::DSAPublicKey::



Collaboration diagram for QCA::DSAPublicKey:



Public Member Functions

- `DSAPublicKey ()`
- `DSAPublicKey (const DLGroup &domain, const BigInteger &y, const QString &provider=QString())`
- `DSAPublicKey (const DSAPrivateKey &k)`
- `DLGroup domain () const`
- `BigInteger y () const`

10.25.1 Detailed Description

Digital Signature Algorithm Public Key.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 QCA::DSAPublicKey::DSAPublicKey ()

Create an empty DSA public key.

10.25.2.2 QCA::DSAPublicKey::DSAPublicKey (const [DLGroup](#) & *domain*, const [BigInteger](#) & *y*, const QString & *provider* = QString ())

Create a DSA public key.

Parameters:

domain the discrete logarithm group to use

y the public random value

provider the provider to use, if a specific provider is required

10.25.2.3 QCA::DSAPublicKey::DSAPublicKey (const [DSAPrivateKey](#) & *k*)

Create a DSA public key from a specified private key.

Parameters:

k the DSA private key to use as the source

10.25.3 Member Function Documentation

10.25.3.1 [DLGroup](#) QCA::DSAPublicKey::domain () const

The discrete logarithm group that is being used.

10.25.3.2 [BigInteger](#) QCA::DSAPublicKey::y () const

The public random value associated with this key.

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

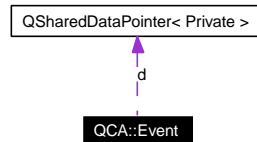
- [qca_publickey.h](#)

10.26 QCA::Event Class Reference

An asynchronous event.

```
#include <qca_core.h>
```

Collaboration diagram for QCA::Event:



Public Types

- enum `Type` { `Password`, `Token` }
- enum `Source` { `KeyStore`, `Data` }
- enum `PasswordStyle` { `StylePassword`, `StylePassphrase`, `StylePIN` }

Public Member Functions

- `Event` ()
- `Event` (const `Event` &from)
- `~Event` ()
- `Event` & `operator=` (const `Event` &from)
- bool `isNull` () const
- `Type` `type` () const
- `Source` `source` () const
- `PasswordStyle` `passwordStyle` () const
- `KeyStoreInfo` `keyStoreInfo` () const
- `KeyStoreEntry` `keyStoreEntry` () const
- `QString` `fileName` () const
- void * `ptr` () const
- void `setPasswordKeyStore` (`PasswordStyle` `pstyle`, const `KeyStoreInfo` &`keyStoreInfo`, const `KeyStoreEntry` &`keyStoreEntry`, void *`ptr`)
- void `setPasswordData` (`PasswordStyle` `pstyle`, const `QString` &`fileName`, void *`ptr`)
- void `setToken` (const `KeyStoreInfo` &`keyStoreInfo`, const `KeyStoreEntry` &`keyStoreEntry`, void *`ptr`)

10.26.1 Detailed Description

An asynchronous event.

Events are produced in response to the library's need for some user intervention, such as entering a pin or password, or inserting a cryptographic token.

`Event` is an abstraction, so you can handle this need in a way that makes sense for your application.

Examples:

`eventhandlerdemo.cpp`, and `keyloader.cpp`.

10.26.2 Member Enumeration Documentation

10.26.2.1 enum [QCA::Event::Type](#)

Type of event

See also:

[type\(\)](#)

Enumerator:

Password Asking for a password, PIN or passphrase.

Token Asking for a token.

10.26.2.2 enum [QCA::Event::Source](#)

Source of the event

Events are associated with access to a [KeyStore](#), or access to a file (or bytearray/stream or equivalent). This tells you the type of source that caused the [Event](#).

See also:

[source\(\)](#)

[fileName\(\)](#) for the name, if [source](#) is [Event::Data](#)

[keyStoreInfo\(\)](#) and [keyStoreEntry\(\)](#) for the keystore and entry, if the [source](#) is [Event::KeyStore](#)

Enumerator:

KeyStore [KeyStore](#) generated the event.

Data File or bytearray generated the event.

10.26.2.3 enum [QCA::Event::PasswordStyle](#)

password variation

If the Type of [Event](#) is Password, PasswordStyle tells you whether it is a PIN, passphrase or password.

See also:

[passwordStyle\(\)](#)

Enumerator:

StylePassword User should be prompted for a "Password".

StylePassphrase User should be prompted for a "Passphrase".

StylePIN User should be prompted for a "PIN".

10.26.3 Constructor & Destructor Documentation

10.26.3.1 [QCA::Event::Event \(\)](#)

Constructor.

10.26.3.2 QCA::Event::Event (const [Event](#) & *from*)

Copy constructor.

Parameters:

from the [Event](#) to copy from

10.26.3.3 QCA::Event::~~Event ()

Destructor.

10.26.4 Member Function Documentation

10.26.4.1 [Event](#)& QCA::Event::operator= (const [Event](#) & *from*)

Assignment operator.

Parameters:

from the [Event](#) to copy from

10.26.4.2 bool QCA::Event::isNull () const

test if this event has been setup correctly

Examples:

[eventhandlerdemo.cpp](#).

10.26.4.3 [Type](#) QCA::Event::type () const

the Type of this event

Examples:

[keyloader.cpp](#).

10.26.4.4 [Source](#) QCA::Event::source () const

the Source of this event

Examples:

[eventhandlerdemo.cpp](#).

10.26.4.5 [PasswordStyle](#) QCA::Event::passwordStyle () const

the style of password required.

This is not meaningful unless the Type is [Event::Password](#).

See also:

[PasswordStyle](#)

10.26.4.6 [KeyStoreInfo](#) QCA::Event::keyStoreInfo () const

The info of the [KeyStore](#) associated with this event.

This is not meaningful unless the Source is [KeyStore](#).

10.26.4.7 [KeyStoreEntry](#) QCA::Event::keyStoreEntry () const

The [KeyStoreEntry](#) associated with this event.

This is not meaningful unless the Source is [KeyStore](#).

10.26.4.8 QString QCA::Event::fileName () const

Name or other identifier for the file or byte array associated with this event.

This is not meaningful unless the Source is Data.

10.26.4.9 void* QCA::Event::ptr () const

opaque data

10.26.4.10 void QCA::Event::setPasswordKeyStore ([PasswordStyle](#) *pstyle*, const [KeyStoreInfo](#) & *keyStoreInfo*, const [KeyStoreEntry](#) & *keyStoreEntry*, void * *ptr*)

Set the values for this [Event](#).

This creates a Password type event, for a keystore.

Parameters:

pstyle the style of information required (e.g. PIN, password or passphrase)

keyStoreInfo info about the keystore that the information is required for

keyStoreEntry the entry in the keystore that the information is required for

ptr opaque data

10.26.4.11 void QCA::Event::setPasswordData ([PasswordStyle](#) *pstyle*, const QString & *fileName*, void * *ptr*)

Set the values for this [Event](#).

This creates a Password type event, for a file.

Parameters:

pstyle the style of information required (e.g. PIN, password or passphrase)

fileName the name of the file (or other identifier) that the information is required for

ptr opaque data

10.26.4.12 void QCA::Event::setToken (const [KeyStoreInfo](#) & *keyStoreInfo*, const [KeyStoreEntry](#) & *keyStoreEntry*, void * *ptr*)

Set the values for this [Event](#).

This creates a Token type event.

Parameters:

keyStoreInfo info about the keystore that the token is required for

keyStoreEntry the entry in the keystore that the token is required for

ptr opaque data

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

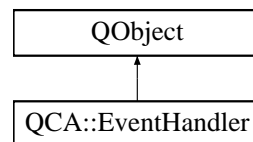
- [qca_core.h](#)

10.27 QCA::EventHandler Class Reference

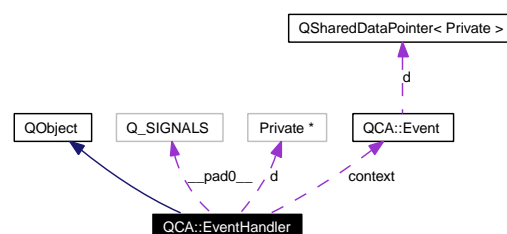
Interface class for password / passphrase / PIN and token handlers.

```
#include <qca_core.h>
```

Inheritance diagram for QCA::EventHandler::



Collaboration diagram for QCA::EventHandler:



Public Member Functions

- [EventHandler](#) ([QObject](#) *parent=0)
- void [start](#) ()
- void [submitPassword](#) (int id, const [SecureArray](#) &password)
- void [tokenOkay](#) (int id)
- void [reject](#) (int id)

Public Attributes

- Q_SIGNALS [__pad0__](#): void eventReady(int id
- Q_SIGNALS const [QCA::Event](#) & [context](#)

Friends

- class [Private](#)

10.27.1 Detailed Description

Interface class for password / passphrase / PIN and token handlers.

This class is used on client side applications to handle the provision of passwords, passphrases and PINs by users, and to indicate that tokens have been correctly inserted.

The concept behind this class is that the library can raise events (typically using [PasswordAsker](#) or [TokenAsker](#)), which may (or may not) be handled by the application using a handler object (that has-a [EventHandler](#), or possibly is-a [EventHandler](#)) that is connected to the `eventReady()` signal.

Examples:

[eventhandlerdemo.cpp](#), and [keyloader.cpp](#).

10.27.2 Constructor & Destructor Documentation

10.27.2.1 QCA::EventHandler::EventHandler (QObject * *parent* = 0)

Constructor.

Parameters:

parent the parent object for this object

10.27.3 Member Function Documentation

10.27.3.1 void QCA::EventHandler::start ()

mandatory function to call after connecting the signal to a slot in your application specific password / passphrase / PIN or token handler

10.27.3.2 void QCA::EventHandler::submitPassword (int *id*, const [SecureArray](#) & *password*)

function to call to return the user provided password, passphrase or PIN.

Parameters:

id the id corresponding to the password request

password the user-provided password, passphrase or PIN.

Note:

the id parameter is the same as that provided in the `eventReady()` signal.

10.27.3.3 void QCA::EventHandler::tokenOkay (int *id*)

function to call to indicate that the token has been inserted by the user.

Parameters:

id the id corresponding to the password request

Note:

the id parameter is the same as that provided in the `eventReady()` signal.

10.27.3.4 void QCA::EventHandler::reject (int *id*)

function to call to indicate that the user declined to provide a password, passphrase, PIN or token.

Parameters:

id the id corresponding to the password request

Note:

the id parameter is the same as that provided in the eventReady() signal.

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

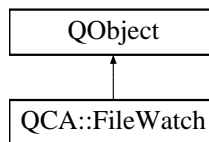
- [qca_core.h](#)

10.28 QCA::FileWatch Class Reference

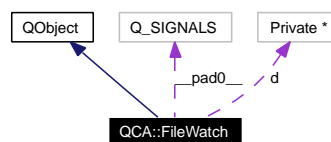
Support class to monitor a file for activity.

```
#include <qca_support.h>
```

Inheritance diagram for QCA::FileWatch::



Collaboration diagram for QCA::FileWatch:



Public Member Functions

- [FileWatch](#) (const **QString** &file=**QString**(), **QObject** *parent=0)
- **QString** [fileName](#) () const
- void [setFileName](#) (const **QString** &file)

Public Attributes

- **Q_SIGNALS** [__pad0__](#): void changed()

Friends

- class **Private**

10.28.1 Detailed Description

Support class to monitor a file for activity.

FileWatch monitors a specified file for any changes. When the file changes, the changed() signal is emitted.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 **QCA::FileWatch::FileWatch** (const **QString** &*file* = **QString** () , **QObject** **parent* = 0)
[explicit]

Standard constructor.

Parameters:

file the name of the file to watch. If not in this object, you can set it using [setFileName\(\)](#)

parent the parent object for this object

10.28.3 Member Function Documentation

10.28.3.1 QString QCA::FileWatch::fileName () const

The name of the file that is being monitored.

10.28.3.2 void QCA::FileWatch::setFileName (const QString &file)

Change the file being monitored.

Parameters:

file the name of the file to monitor

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

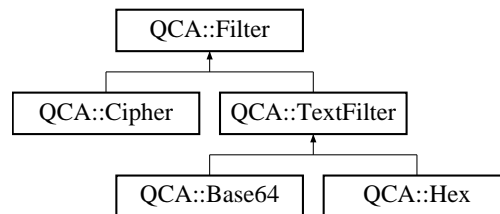
- [qca_support.h](#)

10.29 QCA::Filter Class Reference

General superclass for filtering transformation algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Filter::



Public Member Functions

- virtual void [clear](#) ()=0
- virtual [MemoryRegion](#) [update](#) (const [MemoryRegion](#) &a)=0
- virtual [MemoryRegion](#) [final](#) ()=0
- virtual bool [ok](#) () const =0
- [MemoryRegion](#) [process](#) (const [MemoryRegion](#) &a)

10.29.1 Detailed Description

General superclass for filtering transformation algorithms.

A filtering computation is characterised by having the algorithm take input data in an incremental way, with results delivered for each input, or block of input. Some internal state may be managed, with the transformation completed when [final\(\)](#) is called.

If this seems a bit vague, then you might try deriving your class from a subclass with stronger semantics, or if your [update\(\)](#) function is always returning null results, and everything comes out at [final\(\)](#), try [Buffered-Computation](#).

10.29.2 Member Function Documentation

10.29.2.1 virtual void QCA::Filter::clear () [pure virtual]

Reset the internal state.

Implemented in [QCA::Cipher](#), [QCA::Hex](#), and [QCA::Base64](#).

10.29.2.2 virtual [MemoryRegion](#) QCA::Filter::update (const [MemoryRegion](#) &a) [pure virtual]

Process more data, returning the corresponding filtered version of the data.

Parameters:

a the array containing data to process

Implemented in [QCA::Cipher](#), [QCA::Hex](#), and [QCA::Base64](#).

10.29.2.3 virtual [MemoryRegion](#) QCA::Filter::final () [pure virtual]

Complete the algorithm, returning any additional results.

Implemented in [QCA::Cipher](#), [QCA::Hex](#), and [QCA::Base64](#).

10.29.2.4 virtual bool QCA::Filter::ok () const [pure virtual]

Test if an [update\(\)](#) or [final\(\)](#) call succeeded.

Returns:

true if the previous call succeeded

Implemented in [QCA::Cipher](#), [QCA::Hex](#), and [QCA::Base64](#).

10.29.2.5 [MemoryRegion](#) QCA::Filter::process (const [MemoryRegion](#) & a)

Perform an "all in one" update, returning the result.

This is appropriate if you have all the data in one array - just call process on that array, and you will get back the results of the computation.

Note:

This will invalidate any previous computation using this object.

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

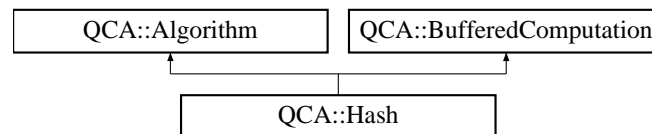
- [qca_core.h](#)

10.30 QCA::Hash Class Reference

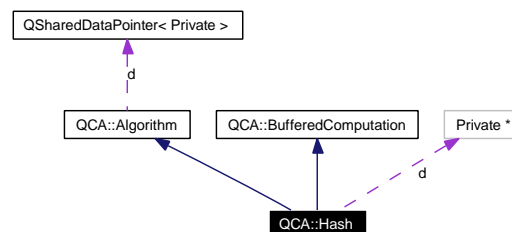
General class for hashing algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Hash::



Collaboration diagram for QCA::Hash:



Public Member Functions

- [Hash](#) (const [QString](#) &type, const [QString](#) &provider=[QString](#)())
- [Hash](#) (const [Hash](#) &from)
- [Hash](#) & [operator=](#) (const [Hash](#) &from)
- [QString](#) [type](#) () const
- virtual void [clear](#) ()
- virtual void [update](#) (const [MemoryRegion](#) &a)
- void [update](#) (const [QByteArray](#) &a)
- void [update](#) (const char *data, int len=-1)
- void [update](#) ([QIODevice](#) *file)
- virtual [MemoryRegion](#) [final](#) ()
- [MemoryRegion](#) [hash](#) (const [MemoryRegion](#) &array)
- [QString](#) [hashToString](#) (const [MemoryRegion](#) &array)

10.30.1 Detailed Description

General class for hashing algorithms.

[Hash](#) is the class for the various hashing algorithms within QCA. SHA256, SHA1 or RIPEMD160 are recommended for new applications, although MD2, MD4, MD5 or SHA0 may be applicable (for interoperability reasons) for some applications.

To perform a hash, you create a [Hash](#) object, call [update\(\)](#) with the data that needs to be hashed, and then call [final\(\)](#), which returns a [QByteArray](#) of the hash result. An example (using the SHA1 hash, with 1000 updates of a 1000 byte string) is shown below:

```

if(!QCA::isSupported("sha1"))
    printf("SHA1 not supported!\n");
else
{
    QByteArray fillerString;
    fillerString.fill('a', 1000);

    QCA::Hash shaHash("sha1");
    for (int i=0; i<1000; i++)
        shaHash.update(fillerString);
    QByteArray hashResult = shaHash.final();
    if ( "34aa973cd4c4daa4f61eeb2bdbad27316534016f" == QCA::arrayToHex(hashResult) )
    {
        printf("big SHA1 is OK\n");
    }
    else
    {
        printf("big SHA1 failed\n");
    }
}

```

If you only have a simple hash requirement - a single string that is fully available in memory at one time - then you may be better off with one of the convenience methods. So, for example, instead of creating a [QCA::Hash](#) object, then doing a single [update\(\)](#) and the [final\(\)](#) call; you could simply call [QCA::Hash\("algoName"\).hash\(\)](#) with the data that you would otherwise have provided to the [update\(\)](#) call.

Examples:

[hashtest.cpp](#), and [md5crypt.cpp](#).

10.30.2 Constructor & Destructor Documentation

10.30.2.1 QCA::Hash::Hash (const QString & *type*, const QString & *provider* = QString ()) [explicit]

Constructor.

Parameters:

- type* label for the type of hash to be created (eg "sha1" or "md2")
- provider* the name of the provider plugin for the subclass (eg "qca-openssl")

10.30.2.2 QCA::Hash::Hash (const Hash & *from*)

Copy constructor.

Parameters:

- from* the [Hash](#) object to copy from

10.30.3 Member Function Documentation

10.30.3.1 Hash& QCA::Hash::operator= (const Hash & *from*)

Assignment operator.

Parameters:

- from* the [Hash](#) object to copy state from

10.30.3.2 QString QCA::Hash::type () const

Return the hash type.

Reimplemented from [QCA::Algorithm](#).

10.30.3.3 virtual void QCA::Hash::clear () [virtual]

Reset a hash, dumping all previous parts of the message.

This method clears (or resets) the hash algorithm, effectively undoing any previous [update\(\)](#) calls. You should use this call if you are re-using a [Hash](#) sub-class object to calculate additional hashes.

Implements [QCA::BufferedComputation](#).

10.30.3.4 virtual void QCA::Hash::update (const MemoryRegion & a) [virtual]

Update a hash, adding more of the message contents to the digest.

The whole message needs to be added using this method before you call [final\(\)](#).

If you find yourself only calling [update\(\)](#) once, you may be better off using a convenience method such as [hash\(\)](#) or [hashToString\(\)](#) instead.

Parameters:

a the byte array to add to the hash

Implements [QCA::BufferedComputation](#).

10.30.3.5 void QCA::Hash::update (const QByteArray & a)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

a the [QByteArray](#) to add to the hash

10.30.3.6 void QCA::Hash::update (const char * data, int len = -1)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This method is provided to assist with code that already exists, and is being ported to QCA.

You are better off passing a [SecureArray](#) (as shown above) if you are writing new code.

Parameters:

data pointer to a char array

len the length of the array. If not specified (or specified as a negative number), the length will be determined with [strlen\(\)](#), which may not be what you want if the array contains a null (0x00) character.

10.30.3.7 void QCA::Hash::update (QIODevice *file)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This allows you to read from a file or other I/O device.

Note that the device must be already open for reading

Parameters:

file an I/O device

If you are trying to calculate the hash of a whole file (and it isn't already open), you might want to use code like this:

```
QFile f( "file.dat" );
if ( fl.open( IO_ReadOnly ) )
{
    QCA::Hash hashObj("sha1");
    hashObj.update( &fl );
    QString output = hashObj.final() ),
}
```

10.30.3.8 virtual MemoryRegion QCA::Hash::final () [virtual]

Finalises input and returns the hash result.

After calling [update\(\)](#) with the required data, the hash results are finalised and produced.

Note that it is not possible to add further data (with [update\(\)](#)) after calling [final\(\)](#), because of the way the hashing works - null bytes are inserted to pad the results up to a fixed size. If you want to reuse the [Hash](#) object, you should call [clear\(\)](#) and start to [update\(\)](#) again.

Implements [QCA::BufferedComputation](#).

10.30.3.9 MemoryRegion QCA::Hash::hash (const MemoryRegion & array)

Hash a byte array, returning it as another byte array

This is a convenience method that returns the hash of a [SecureArray](#).

```
SecureArray sampleArray(3);
sampleArray.fill('a');
SecureArray outputArray = QCA::Hash("md2")::hash(sampleArray);
```

Parameters:

array the QByteArray to hash

If you need more flexibility (e.g. you are constructing a large byte array object just to pass it to [hash\(\)](#), then consider creating an [Hash](#) object, and then calling [update\(\)](#) and [final\(\)](#).

10.30.3.10 QString QCA::Hash::hashToString (const MemoryRegion & array)

Hash a byte array, returning it as a printable string

This is a convenience method that returns the hash of a QSecureArray as a hexadecimal representation encoded in a [QString](#).

Parameters:

array the `QByteArray` to hash

If you need more flexibility, you can create a [Hash](#) object, call [Hash::update\(\)](#) as required, then call [Hash::final\(\)](#), before using the static [arrayToHex\(\)](#) method.

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

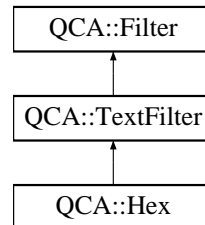
- [qca_basic.h](#)

10.31 QCA::Hex Class Reference

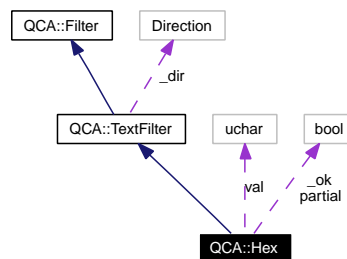
Hexadecimal encoding / decoding.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Hex::



Collaboration diagram for QCA::Hex:



Public Member Functions

- [Hex](#) ([Direction](#) dir=Encode)
- virtual void [clear](#) ()
- virtual [MemoryRegion](#) [update](#) (const [MemoryRegion](#) &a)
- virtual [MemoryRegion](#) [final](#) ()
- virtual bool [ok](#) () const

10.31.1 Detailed Description

Hexadecimal encoding / decoding.

Examples:

[hextest.cpp](#), and [randomtest.cpp](#).

10.31.2 Constructor & Destructor Documentation

10.31.2.1 QCA::Hex::Hex ([Direction](#) *dir* = Encode)

Standard constructor.

Parameters:

dir the Direction that should be used.

Note:

The direction can be changed using the [setup\(\)](#) call.

10.31.3 Member Function Documentation

10.31.3.1 virtual void QCA::Hex::clear () [virtual]

Reset the internal state.

This is useful to reuse an existing [Hex](#) object

Implements [QCA::Filter](#).

10.31.3.2 virtual [MemoryRegion](#) QCA::Hex::update (const [MemoryRegion](#) & a) [virtual]

Process more data, returning the corresponding encoded or decoded (depending on the Direction set in the constructor or [setup\(\)](#) call) representation.

If you find yourself with code that only calls this method once, you might be better off using [encode\(\)](#) or [decode\(\)](#). Similarly, if the data is really a string, you might be better off using [arrayToString\(\)](#), [encodeString\(\)](#), [stringToArray\(\)](#) or [decodeString\(\)](#).

Parameters:

a the array containing data to process

Implements [QCA::Filter](#).

10.31.3.3 virtual [MemoryRegion](#) QCA::Hex::final () [virtual]

Complete the algorithm.

Returns:

any remaining output. Because of the way hexadecimal encoding works, this will return a zero length array - any output will have been returned from the [update\(\)](#) call.

Implements [QCA::Filter](#).

10.31.3.4 virtual bool QCA::Hex::ok () const [virtual]

Test if an [update\(\)](#) or [final\(\)](#) call succeeded.

Returns:

true if the previous call succeeded

Implements [QCA::Filter](#).

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

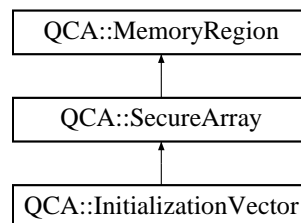
- [qca_textfilter.h](#)

10.32 QCA::InitializationVector Class Reference

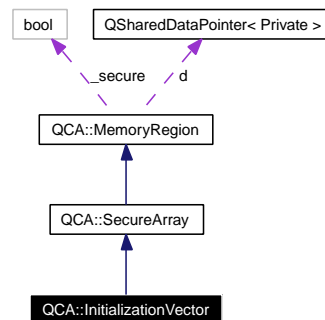
Container for initialisation vectors and nonces.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::InitializationVector::



Collaboration diagram for QCA::InitializationVector:



Public Member Functions

- [InitializationVector](#) ()
- [InitializationVector](#) (int size)
- [InitializationVector](#) (const [SecureArray](#) &a)
- [InitializationVector](#) (const [QByteArray](#) &a)

10.32.1 Detailed Description

Container for initialisation vectors and nonces.

Examples:

[ciphertest.cpp](#).

10.32.2 Constructor & Destructor Documentation

10.32.2.1 QCA::InitializationVector::InitializationVector ()

Construct an empty (zero length) initisation vector.

10.32.2.2 QCA::InitializationVector::InitializationVector (int *size*)

Construct an initialisation vector of the specified size.

Parameters:

size the length of the initialisation vector, in bytes

10.32.2.3 QCA::InitializationVector::InitializationVector (const SecureArray & *a*)

Construct an initialisation vector from a provided byte array.

Parameters:

a the byte array to copy

10.32.2.4 QCA::InitializationVector::InitializationVector (const QByteArray & *a*)

Construct an initialisation vector from a provided byte array.

Parameters:

a the byte array to copy

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

- [qca_core.h](#)

10.33 QCA::Initializer Class Reference

Convenience method for initialising and cleaning up QCA.

```
#include <QtCrypto>
```

Public Member Functions

- [Initializer](#) ([MemoryMode](#) m=Practical, int prealloc=64)

10.33.1 Detailed Description

Convenience method for initialising and cleaning up QCA.

To ensure that [QCA](#) is properly initialised and cleaned up, it is convenient to create an [Initializer](#) object, and let it go out of scope at the end of QCA usage.

Examples:

[aes-cmac.cpp](#), [base64test.cpp](#), [certtest.cpp](#), [ciphertest.cpp](#), [cmsexample.cpp](#), [eventhandlerdemo.cpp](#), [hashtest.cpp](#), [hextest.cpp](#), [keyloader.cpp](#), [mactest.cpp](#), [main.cpp](#), [md5crypt.cpp](#), [providertest.cpp](#), [publickeyexample.cpp](#), [randomtest.cpp](#), [rsatest.cpp](#), [saslservertest.cpp](#), [sasltest.cpp](#), [sslservtest.cpp](#), and [ssltest.cpp](#).

10.33.2 Constructor & Destructor Documentation

10.33.2.1 QCA::Initializer::Initializer ([MemoryMode](#) *m* = Practical, int *prealloc* = 64) [explicit]

Standard constructor.

Parameters:

- m* the [MemoryMode](#) to use for secure memory
- prealloc* the amount of secure memory to pre-allocate, in units of 1024 bytes (1K).

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

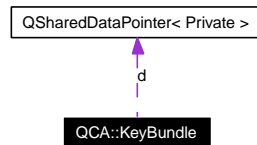
- [qca_core.h](#)

10.34 QCA::KeyBundle Class Reference

[Certificate](#) chain and private key pair.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::KeyBundle:



Public Member Functions

- [KeyBundle](#) ()
- [KeyBundle](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase=[SecureArray](#)())
- [KeyBundle](#) (const [KeyBundle](#) &from)
- [KeyBundle](#) & operator= (const [KeyBundle](#) &from)
- bool [isNull](#) () const
- [QString](#) [name](#) () const
- [CertificateChain](#) [certificateChain](#) () const
- [PrivateKey](#) [privateKey](#) () const
- void [setName](#) (const [QString](#) &s)
- void [setCertificateChainAndKey](#) (const [CertificateChain](#) &c, const [PrivateKey](#) &key)
- [QByteArray](#) [toArray](#) (const [SecureArray](#) &passphrase, const [QString](#) &provider=[QString](#)()) const
- bool [toFile](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase, const [QString](#) &provider=[QString](#)()) const

Static Public Member Functions

- static [KeyBundle](#) [fromArray](#) (const [QByteArray](#) &a, const [SecureArray](#) &passphrase=[SecureArray](#)(), [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [KeyBundle](#) [fromFile](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase=[SecureArray](#)(), [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

10.34.1 Detailed Description

[Certificate](#) chain and private key pair.

[KeyBundle](#) is essentially a convenience class that holds a certificate chain and an associated private key. This class has a number of methods that make it particularly suitable for accessing a PKCS12 (.p12) format file, however it can be used as just a container for a [Certificate](#), its associated [PrivateKey](#) and optionally additional X.509 [Certificate](#) that form a chain.

For more information on PKCS12 "Personal Information Exchange Syntax Standard", see <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>.

10.34.2 Constructor & Destructor Documentation

10.34.2.1 QCA::KeyBundle::KeyBundle ()

Create an empty [KeyBundle](#).

10.34.2.2 QCA::KeyBundle::KeyBundle (const QString & *fileName*, const SecureArray & *passphrase* = SecureArray ()) [explicit]

Create a [KeyBundle](#) from a PKCS12 (.p12) encoded file.

This constructor requires appropriate plugin (provider) support. You must check for the "pkcs12" feature before using this constructor.

Parameters:

fileName the name of the file to read from

passphrase the passphrase that is applicable to the file

See also:

[fromFile](#) for a more flexible version of the same capability.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.34.2.3 QCA::KeyBundle::KeyBundle (const KeyBundle & *from*)

Standard copy constructor.

Parameters:

from the [KeyBundle](#) to use as source

10.34.3 Member Function Documentation

10.34.3.1 KeyBundle& QCA::KeyBundle::operator= (const KeyBundle & *from*)

Standard assignment operator.

Parameters:

from the [KeyBundle](#) to use as source

10.34.3.2 bool QCA::KeyBundle::isNull () const

Test if this key is empty (null).

10.34.3.3 QString QCA::KeyBundle::name () const

The name associated with this key.

This is also known as the "friendly name", and if present, is typically suitable to be displayed to the user.

See also:

[setName](#)

10.34.3.4 CertificateChain QCA::KeyBundle::certificateChain () const

The public certificate part of this bundle.

See also:

[setCertificateChainAndKey](#)

10.34.3.5 PrivateKey QCA::KeyBundle::privateKey () const

The private key part of this bundle.

See also:

[setCertificateChainAndKey](#)

10.34.3.6 void QCA::KeyBundle::setName (const QString & s)

Specify the name of this bundle.

Parameters:

s the name to use

10.34.3.7 void QCA::KeyBundle::setCertificateChainAndKey (const CertificateChain & c, const PrivateKey & key)

Set the public certificate and private key.

Parameters:

c the [CertificateChain](#) containing the public part of the Bundle

key the private key part of the Bundle

See also:

[privateKey](#), [certificateChain](#) for getters

10.34.3.8 QByteArray QCA::KeyBundle::toArray (const SecureArray & passphrase, const QString & provider = QString ()) const

Export the key bundle to an array in PKCS12 format.

This method requires appropriate plugin (provider) support - you must check for the "pkcs12" feature, as shown below.

```
if( QCA::isSupported("pkcs12") )
{
    // can use I/O
    byteArray = bundle.toArray( "pass phrase" );
}
else
{
    // not possible to use I/O
}
```

Parameters:

passphrase the passphrase to use to protect the bundle
provider the provider to use, if a specific provider is required

10.34.3.9 bool QCA::KeyBundle::toFile (const QString & fileName, const SecureArray & passphrase, const QString & provider = QString ()) const

Export the key bundle to a file in PKCS12 (.p12) format.

This method requires appropriate plugin (provider) support - you must check for the "pkcs12" feature, as shown below.

```
if( QCA::isSupported("pkcs12") )
{
    // can use I/O
    bool result = bundle.toFile( filename, "pass phrase" );
}
else
{
    // not possible to use I/O
}
```

Parameters:

fileName the name of the file to save to
passphrase the passphrase to use to protect the bundle
provider the provider to use, if a specific provider is required

10.34.3.10 static KeyBundle QCA::KeyBundle::fromArray (const QByteArray & a, const SecureArray & passphrase = SecureArray(), ConvertResult * result = 0, const QString & provider = QString ()) [static]

Import the key bundle from an array in PKCS12 format.

This method requires appropriate plugin (provider) support - you must check for the "pkcs12" feature, as shown below.

```

if( QCA::isSupported("pkcs12") )
{
    // can use I/O
    bundle = QCA::KeyBundle::fromArray( array, "pass phrase" );
}
else
{
    // not possible to use I/O
}

```

Parameters:

- a* the array to import from
- passphrase* the passphrase for the encoded bundle
- result* pointer to the result of the import process
- provider* the provider to use, if a specific provider is required

See also:

[QCA::KeyLoader](#) for an asynchronous loader approach.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.34.3.11 static [KeyBundle](#) QCA::KeyBundle::fromFile (const QString & *fileName*, const [SecureArray](#) & *passphrase* = [SecureArray](#) (), [ConvertResult](#) * *result* = 0, const QString & *provider* = QString ()) [static]

Import the key bundle from a file in PKCS12 (.p12) format.

This method requires appropriate plugin (provider) support - you must check for the "pkcs12" feature, as shown below.

```

if( QCA::isSupported("pkcs12") )
{
    // can use I/O
    bundle = QCA::KeyBundle::fromFile( filename, "pass phrase" );
}
else
{
    // not possible to use I/O
}

```

Parameters:

- fileName* the name of the file to read from
- passphrase* the passphrase for the encoded bundle
- result* pointer to the result of the import process
- provider* the provider to use, if a specific provider is required

See also:

[QCA::KeyLoader](#) for an asynchronous loader approach.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

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

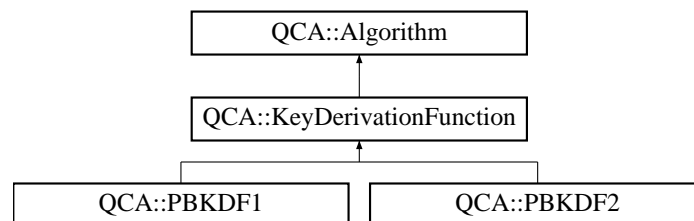
- [qca_cert.h](#)

10.35 QCA::KeyDerivationFunction Class Reference

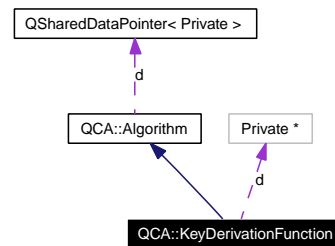
General superclass for key derivation algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::KeyDerivationFunction::



Collaboration diagram for QCA::KeyDerivationFunction:



Public Member Functions

- [KeyDerivationFunction](#) (const [KeyDerivationFunction](#) &from)
- [KeyDerivationFunction](#) & operator= (const [KeyDerivationFunction](#) &from)
- [SymmetricKey](#) makeKey (const [SecureArray](#) &secret, const [InitializationVector](#) &salt, unsigned int keyLength, unsigned int iterationCount)

Static Public Member Functions

- static [QString](#) withAlgorithm (const [QString](#) &kdfType, const [QString](#) &algType)

Protected Member Functions

- [KeyDerivationFunction](#) (const [QString](#) &type, const [QString](#) &provider)

10.35.1 Detailed Description

General superclass for key derivation algorithms.

KeyDerivationFunction is a superclass for the various key derivation function algorithms within QCA. You should not need to use it directly unless you are adding another key derivation capability to QCA - you should be using a sub-class. [PBKDF2](#) using SHA1 is recommended for new applications.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 QCA::KeyDerivationFunction::KeyDerivationFunction (const [KeyDerivationFunction](#) & *from*)

Standard copy constructor.

10.35.2.2 QCA::KeyDerivationFunction::KeyDerivationFunction (const QString & *type*, const QString & *provider*) [protected]

Special constructor for subclass initialisation.

10.35.3 Member Function Documentation

10.35.3.1 [KeyDerivationFunction](#)& QCA::KeyDerivationFunction::operator= (const [KeyDerivationFunction](#) & *from*)

Assignment operator.

Copies the state (including key) from one [KeyDerivationFunction](#) to another

10.35.3.2 [SymmetricKey](#) QCA::KeyDerivationFunction::makeKey (const [SecureArray](#) & *secret*, const [InitializationVector](#) & *salt*, unsigned int *keyLength*, unsigned int *iterationCount*)

Generate the key from a specified secret and salt value.

Note:

key length is ignored for some functions

Parameters:

secret the secret (password or passphrase)

salt the salt to use

keyLength the length of key to return

iterationCount the number of iterations to perform

Returns:

the derived key

10.35.3.3 static QString QCA::KeyDerivationFunction::withAlgorithm (const QString & *kdfType*, const QString & *algType*) [static]

Construct the name of the algorithm.

You can use this to build a standard name string. You probably only need this method if you are creating a new subclass.

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

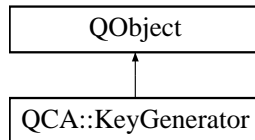
- [qca_basic.h](#)

10.36 QCA::KeyGenerator Class Reference

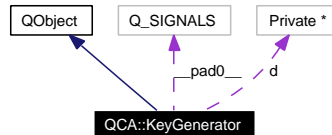
Class for generating asymmetric key pairs.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::KeyGenerator::



Collaboration diagram for QCA::KeyGenerator:



Public Member Functions

- [KeyGenerator](#) (**QObject** *parent=0)
- bool [blockingEnabled](#) () const
- void [setBlockingEnabled](#) (bool b)
- bool [isBusy](#) () const
- [PrivateKey](#) [createRSA](#) (int bits, int exp=65537, const **QString** &provider=**QString**())
- [PrivateKey](#) [createDSA](#) (const [DLGroup](#) &domain, const **QString** &provider=**QString**())
- [PrivateKey](#) [createDH](#) (const [DLGroup](#) &domain, const **QString** &provider=**QString**())
- [PrivateKey](#) [key](#) () const
- [DLGroup](#) [createDLGroup](#) ([QCA::DLGroupSet](#) set, const **QString** &provider=**QString**())
- [DLGroup](#) [dlGroup](#) () const

Public Attributes

- **Q_SIGNALS** [__pad0__](#): void finished()

Friends

- class **Private**

10.36.1 Detailed Description

Class for generating asymmetric key pairs.

This class is used for generating asymmetric keys (public/private key pairs)

Examples:

[rsatest.cpp](#).

10.36.2 Constructor & Destructor Documentation

10.36.2.1 QCA::KeyGenerator::KeyGenerator (QObject * *parent* = 0)

Create a new key generator.

Parameters:

parent the parent object, if applicable

10.36.3 Member Function Documentation

10.36.3.1 bool QCA::KeyGenerator::blockingEnabled () const

Test whether the key generator is set to operate in blocking mode, or not.

Returns:

true if the key generator is in blocking mode

See also:

[setBlockingEnabled](#)

10.36.3.2 void QCA::KeyGenerator::setBlockingEnabled (bool *b*)

Set whether the key generator is in blocking mode, nor not.

Parameters:

b if true, the key generator will be set to operate in blocking mode, otherwise it will operate in non-blocking mode

See also:

[blockingEnabled\(\)](#)

10.36.3.3 bool QCA::KeyGenerator::isBusy () const

Test if the key generator is currently busy, or not.

Returns:

true if the key generator is busy generating a key already

10.36.3.4 [PrivateKey](#) QCA::KeyGenerator::createRSA (int *bits*, int *exp* = 65537, const QString & *provider* = QString ())

Generate an RSA key of the specified length.

This method creates both the public key and corresponding private key. You almost certainly want to extract the public key part out - see [PKey::toPublicKey](#) for an easy way.

Key length is a tricky judgment - using less than 2048 is probably being too liberal for long term use. Don't use less than 1024 without serious analysis.

Parameters:

bits the length of key that is required

exp the exponent - typically 3, 17 or 65537

provider the name of the provider to use, if a particular provider is required

10.36.3.5 PrivateKey QCA::KeyGenerator::createDSA (const DLGroup & domain, const QString & provider = QString ())

Generate a DSA key.

This method creates both the public key and corresponding private key. You almost certainly want to extract the public key part out - see [PKey::toPublicKey](#) for an easy way.

Parameters:

domain the discrete logarithm group that this key should be generated from

provider the name of the provider to use, if a particular provider is required

Note:

Not every [DLGroup](#) makes sense for DSA. You should use one of DSA_512, DSA_768 and DSA_1024.

10.36.3.6 PrivateKey QCA::KeyGenerator::createDH (const DLGroup & domain, const QString & provider = QString ())

Generate a Diffie-Hellman key.

This method creates both the public key and corresponding private key. You almost certainly want to extract the public key part out - see [PKey::toPublicKey](#) for an easy way.

Parameters:

domain the discrete logarithm group that this key should be generated from

provider the name of the provider to use, if a particular provider is required

Note:

For compatibility, you should use one of the IETF_ groupsets as the domain argument.

10.36.3.7 PrivateKey QCA::KeyGenerator::key () const

Return the last generated key.

This is really only useful when you are working with non-blocking key generation

10.36.3.8 DLGroup QCA::KeyGenerator::createDLGroup (QCA::DLGroupSet set, const QString & provider = QString ())

Create a new discrete logarithm group.

Parameters:

set the set of discrete logarithm parameters to generate from

provider the name of the provider to use, if a particular provider is required.

10.36.3.9 [DLGroup](#) QCA::KeyGenerator::dlGroup () const

The current discrete logarithm group.

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

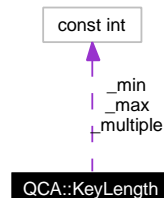
- [qca_publickey.h](#)

10.37 QCA::KeyLength Class Reference

Simple container for acceptable key lengths.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::KeyLength:



Public Member Functions

- [KeyLength](#) (int min, int max, int multiple)
- int [minimum](#) () const
- int [maximum](#) () const
- int [multiple](#) () const

10.37.1 Detailed Description

Simple container for acceptable key lengths.

The [KeyLength](#) specifies the minimum and maximum byte sizes allowed for a key, as well as a "multiple" which the key size must evenly divide into.

As an example, if the key can be 4, 8 or 12 bytes, you can express this as

```
KeyLength keyLen( 4, 12, 4 );
```

If you want to express a [KeyLength](#) that takes any number of bytes (including zero), you may want to use

```
#include<limits>
KeyLength( 0, std::numeric_limits<int>::max(), 1 );
```

Examples:

[aes-cmac.cpp](#).

10.37.2 Constructor & Destructor Documentation

10.37.2.1 QCA::KeyLength::KeyLength (int *min*, int *max*, int *multiple*) `[inline]`

Construct a KeyLength object.

Parameters:

- min*** the minimum length of the key, in bytes
- max*** the maximum length of the key, in bytes
- multiple*** the number of bytes that the key must be a multiple of.

10.37.3 Member Function Documentation

10.37.3.1 `int QCA::KeyLength::minimum () const` [inline]

Obtain the minimum length for the key, in bytes.

10.37.3.2 `int QCA::KeyLength::maximum () const` [inline]

Obtain the maximum length for the key, in bytes.

10.37.3.3 `int QCA::KeyLength::multiple () const` [inline]

Return the number of bytes that the key must be a multiple of.

If this is one, then anything between minimum and maximum (inclusive) is acceptable.

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

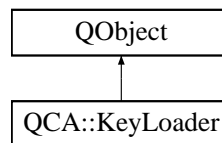
- [qca_core.h](#)

10.38 QCA::KeyLoader Class Reference

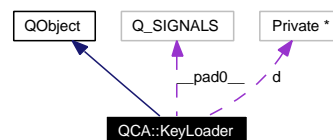
Asynchronous private key loader.

```
#include <qca_cert.h>
```

Inheritance diagram for QCA::KeyLoader::



Collaboration diagram for QCA::KeyLoader:



Public Member Functions

- [KeyLoader](#) ([QObject](#) *parent=0)
- void [loadPrivateKeyFromPEMFile](#) (const [QString](#) &fileName)
- void [loadPrivateKeyFromPEM](#) (const [QString](#) &s)
- void [loadPrivateKeyFromDER](#) (const [SecureArray](#) &a)
- void [loadKeyBundleFromFile](#) (const [QString](#) &fileName)
- void [loadKeyBundleFromArray](#) (const [QByteArray](#) &a)
- [ConvertResult](#) [convertResult](#) () const
- [PrivateKey](#) [privateKey](#) () const
- [KeyBundle](#) [keyBundle](#) () const

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void finished()

Friends

- class [Private](#)

10.38.1 Detailed Description

Asynchronous private key loader.

GUI applications generally must use [KeyLoader](#) to load private keys. This is because the synchronous private key loading functions, for example [QCA::PrivateKey::fromPEMFile\(\)](#), cannot be used within the same thread as an [EventHandler](#), and most GUI applications will use [EventHandler](#) from the main thread.

[KeyLoader](#) does not have this problem. It can be used from any thread, including the same thread as [EventHandler](#).

The [KeyLoader](#) class allows you to asynchronously load stand-alone private keys ([QCA::PrivateKey](#)) or private keys with a certificate ([QCA::KeyBundle](#)) with a signal that advises of completion.

To use this class to load a [PrivateKey](#), you create a [KeyLoader](#) object then use one of the `loadPrivateKeyFrom...`() functions, depending on the format for your key. These functions return immediately. When you get the `finished()` signal, you can check that the loading operation succeeded (using `convertResult()`) and then obtain the [PrivateKey](#) using the `privateKey()` function.

The same process applies for loading a [KeyBundle](#), except that you use either `loadKeyBundleFromFile()` or `loadKeyBundleFromArray()` instead of the `loadPrivateKeyFrom...`() function, and use `keyBundle()` instead of `privateKey()`.

The loader may need a passphrase to complete the loading of the key or key bundle. You should use the [QCA::EventHandler](#) class to ensure that you deal with this correctly.

Note:

QCA also provides synchronous private key loading using [QCA::PrivateKey::fromPEMFile\(\)](#), [QCA::PrivateKey::fromPEM\(\)](#) and [QCA::PrivateKey::fromDER\(\)](#). QCA provides synchronous key bundle loading using [QCA::KeyBundle::fromArray\(\)](#) and [QCA::KeyBundle::fromFile\(\)](#).

Examples:

[keyloader.cpp](#).

10.38.2 Constructor & Destructor Documentation

10.38.2.1 QCA::KeyLoader::KeyLoader (QObject * *parent* = 0)

Create a [KeyLoader](#) object.

Parameters:

parent the parent object for this object

10.38.3 Member Function Documentation

10.38.3.1 void QCA::KeyLoader::loadPrivateKeyFromPEMFile (const QString & *fileName*)

Initiate an asynchronous loading of a [PrivateKey](#) from a PEM format file.

This function will return immediately.

Parameters:

fileName the name of the file (and path, if necessary) to load the key from

10.38.3.2 void QCA::KeyLoader::loadPrivateKeyFromPEM (const QString & *s*)

Initiate an asynchronous loading of a [PrivateKey](#) from a PEM format string.

This function will return immediately.

Parameters:

s the string containing the PEM formatted key

10.38.3.3 void QCA::KeyLoader::loadPrivateKeyFromDER (const [SecureArray](#) & *a*)

Initiate an asynchronous loading of a [PrivateKey](#) from a DER format array.

This function will return immediately.

Parameters:

a the array containing the DER formatted key

10.38.3.4 void QCA::KeyLoader::loadKeyBundleFromFile (const QString & *fileName*)

Initiate an asynchronous loading of a [KeyBundle](#) from a file.

This function will return immediately.

Parameters:

fileName the name of the file (and path, if necessary) to load the key bundle from

10.38.3.5 void QCA::KeyLoader::loadKeyBundleFromArray (const QByteArray & *a*)

Initiate an asynchronous loading of a [KeyBundle](#) from an array.

This function will return immediately.

Parameters:

a the array containing the key bundle

10.38.3.6 [ConvertResult](#) QCA::KeyLoader::convertResult () const

The result of the loading process.

This is not valid until the finished() signal has been emitted.

10.38.3.7 [PrivateKey](#) QCA::KeyLoader::privateKey () const

The private key that has been loaded.

This is only valid if [loadPrivateKeyFromPEMFile\(\)](#), [loadPrivateKeyFromPEM\(\)](#) or [loadPrivateKeyFromDER\(\)](#) has been used, the load has completed (that is, finished() has been emitted), and the conversion succeeded (that is, [convertResult\(\)](#) returned ConvertGood).

10.38.3.8 [KeyBundle](#) QCA::KeyLoader::keyBundle () const

The key bundle that has been loaded.

This is only valid if [loadKeyBundleFromFile\(\)](#) or [loadKeyBundleFromArray\(\)](#) has been used, the load has completed (that is, finished() has been emitted), and the conversion succeeded (that is, [convertResult\(\)](#) returned ConvertGood).

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

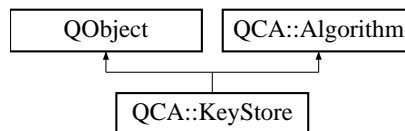
- [qca_cert.h](#)

10.39 QCA::KeyStore Class Reference

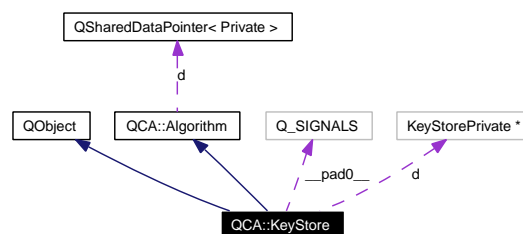
General purpose key storage object.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::KeyStore::



Collaboration diagram for QCA::KeyStore:



Public Types

- enum `Type` {
 System, User, Application, SmartCard,
 PGPKeyring }

Public Member Functions

- `KeyStore` (const **QString** &id, `KeyStoreManager` *keyStoreManager)
- `bool isValid` () const
- `Type type` () const
- `QString name` () const
- `QString id` () const
- `bool isReadOnly` () const
- `void startAsynchronousMode` ()
- `QList< KeyStoreEntry > entryList` () const
- `bool holdsTrustedCertificates` () const
- `bool holdsIdentities` () const
- `bool holdsPGPPublicKeys` () const
- `QString writeEntry` (const `KeyBundle` &kb)
- `QString writeEntry` (const `Certificate` &cert)
- `QString writeEntry` (const `CRL` &crl)
- `QString writeEntry` (const `PGPKey` &key)
- `bool removeEntry` (const **QString** &id)

- void [unavailable](#) ()
- void [entryWritten](#) (const **QString** &entryId)
- void [entryRemoved](#) (bool success)

Public Attributes

- Q_SIGNALS __pad0__: void updated()

Friends

- class **KeyStorePrivate**
- class **KeyStoreManagerPrivate**

10.39.1 Detailed Description

General purpose key storage object.

Examples of use of this are:

- systemstore: System TrustedCertificates
- accepted self-signed: Application TrustedCertificates
- apple keychain: User Identities
- smartcard: SmartCard Identities
- gnupg: GPGKeyring Identities,PGPPublicKeys

Note:

- there can be multiple [KeyStore](#) objects referring to the same id
- when a [KeyStore](#) is constructed, it refers to a given id (deviceId) and internal contextId. if the context goes away, the [KeyStore](#) becomes invalid ([isValid\(\)](#) == false), and [unavailable\(\)](#) is emitted. even if the device later reappears, the [KeyStore](#) remains invalid. a new [KeyStore](#) will have to be created to use the device again.

10.39.2 Member Enumeration Documentation

10.39.2.1 enum [QCA::KeyStore::Type](#)

The type of keystore.

Enumerator:

System objects such as root certificates

User objects such as Apple Keychain, KDE Wallet

Application for caching accepted self-signed certificates

SmartCard for smartcards

PGPKeyring for a PGP keyring

10.39.3 Constructor & Destructor Documentation

10.39.3.1 `QCA::KeyStore::KeyStore (const QString & id, KeyStoreManager * keyStoreManager)`

Obtain a specific [KeyStore](#).

Parameters:

id the identification for the key store

keyStoreManager the parent manager for this keystore

10.39.4 Member Function Documentation

10.39.4.1 `bool QCA::KeyStore::isValid () const`

Check if this [KeyStore](#) is valid.

Returns:

true if the [KeyStore](#) is valid

10.39.4.2 `Type QCA::KeyStore::type () const`

The [KeyStore](#) Type.

Reimplemented from [QCA::Algorithm](#).

10.39.4.3 `QString QCA::KeyStore::name () const`

The name associated with the [KeyStore](#).

10.39.4.4 `QString QCA::KeyStore::id () const`

The ID associated with the [KeyStore](#).

10.39.4.5 `bool QCA::KeyStore::isReadOnly () const`

Test if the [KeyStore](#) is writeable or not.

Returns:

true if the [KeyStore](#) is read-only

10.39.4.6 `void QCA::KeyStore::startAsynchronousMode ()`

Turns on asynchronous mode for this [KeyStore](#) instance.

Normally, [entryList\(\)](#) and [writeEntry\(\)](#) are blocking calls. However, if [startAsynchronousMode\(\)](#) is called, then these functions will return immediately. [entryList\(\)](#) will return with the latest known entries, or an empty list if none are known yet (in this mode, [updated\(\)](#) will be emitted once the initial entries are known, even if the store has not actually been altered). [writeEntry\(\)](#) will always return an empty string, and the [entryWritten\(\)](#) signal indicates the result of a write.

10.39.4.7 QList<KeyStoreEntry> QCA::KeyStore::entryList () const

A list of the [KeyStoreEntry](#) objects in this store.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#) (this is not a concern if asynchronous mode is enabled).

See also:

[startAsynchronousMode](#)

10.39.4.8 bool QCA::KeyStore::holdsTrustedCertificates () const

test if the [KeyStore](#) holds trusted certificates (and CRLs)

10.39.4.9 bool QCA::KeyStore::holdsIdentities () const

test if the [KeyStore](#) holds identities (eg [KeyBundle](#) or [PGPSecretKey](#))

10.39.4.10 bool QCA::KeyStore::holdsPGPPublicKeys () const

test if the [KeyStore](#) holds [PGPPublicKey](#) objects

10.39.4.11 QString QCA::KeyStore::writeEntry (const [KeyBundle](#) & kb)

Add a entry to the [KeyStore](#).

Returns the entryId of the written entry or an empty string on failure.

Parameters:

kb the [KeyBundle](#) to add to the [KeyStore](#)

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#) (this is not a concern if asynchronous mode is enabled).

See also:

[startAsynchronousMode](#)

10.39.4.12 QString QCA::KeyStore::writeEntry (const [Certificate](#) & cert)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

cert the [Certificate](#) to add to the [KeyStore](#)

10.39.4.13 QString QCA::KeyStore::writeEntry (const CRL & *crl*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

crl the CRL to add to the KeyStore

10.39.4.14 QString QCA::KeyStore::writeEntry (const PGPKey & *key*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

key the PGPKey to add to the KeyStore

Returns:

a ref to the key in the keyring

10.39.4.15 bool QCA::KeyStore::removeEntry (const QString & *id*)

Delete the a specified KeyStoreEntry from this KeyStore.

Parameters:

id the ID for the entry to be deleted

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an EventHandler (this is not a concern if asynchronous mode is enabled).

See also:

[startAsynchronousMode](#)

10.39.4.16 void QCA::KeyStore::unavailable ()

Emitted when the KeyStore becomes unavailable.

10.39.4.17 void QCA::KeyStore::entryWritten (const QString & *entryId*)

Emitted when an entry has been written, in asynchronous mode.

entryId is the newly written entry id on success, or an empty string if the write failed.

10.39.4.18 void QCA::KeyStore::entryRemoved (bool *success*)

Emitted when an entry has been removed, in asynchronous mode.

success indicates if the removal succeeded or not.

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

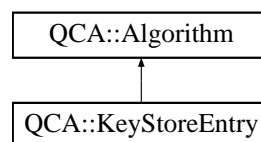
- [qca_keystore.h](#)

10.40 QCA::KeyStoreEntry Class Reference

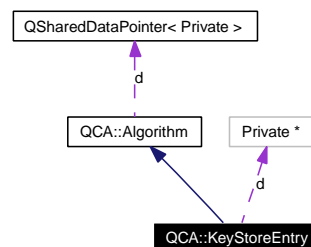
Single entry in a [KeyStore](#).

```
#include <QtCrypto>
```

Inheritance diagram for QCA::KeyStoreEntry::



Collaboration diagram for QCA::KeyStoreEntry:



Public Types

- enum [Type](#) {
TypeKeyBundle, **TypeCertificate**, **TypeCRL**, **TypePGPSecretKey**,
TypePGPPublicKey }

Public Member Functions

- [KeyStoreEntry](#) ()
- [KeyStoreEntry](#) (const **QString** &serialized)
- [KeyStoreEntry](#) (const [KeyStoreEntry](#) &from)
- [KeyStoreEntry](#) & **operator=** (const [KeyStoreEntry](#) &from)
- bool [isNull](#) () const
- bool [isAvailable](#) () const
- bool [isAccessible](#) () const
- [Type](#) [type](#) () const
- **QString** [name](#) () const
- **QString** [id](#) () const
- **QString** [storeName](#) () const
- **QString** [storeId](#) () const
- **QString** [toString](#) () const
- [KeyBundle](#) [keyBundle](#) () const
- [Certificate](#) [certificate](#) () const
- [CRL](#) [crl](#) () const

- [PGPKey](#) [pgpSecretKey](#) () const
- [PGPKey](#) [pgpPublicKey](#) () const
- bool [ensureAvailable](#) ()
- bool [ensureAccess](#) ()

Static Public Member Functions

- static [KeyStoreEntry](#) [fromString](#) (const [QString](#) &serialized)

Friends

- class [KeyStoreTracker](#)

10.40.1 Detailed Description

Single entry in a [KeyStore](#).

This is a container for any kind of object in a [KeyStore](#) (such as PGP keys, or X.509 certificates / private keys).

[KeyStoreEntry](#) objects are obtained through [KeyStore](#) or loaded from a serialized string format. The latter method requires a [KeyStoreEntry](#) obtained through [KeyStore](#) to be serialized for future loading. For example:

```
QString str = someKeyStoreEntry.toString();
[ app saves str to disk ]
[ app quits ]
...
[ app launches ]
[ app reads str from disk ]
KeyStoreEntry entry(str);
printf("Entry name: [%s]\n", qPrintable(entry.name()));
```

[KeyStoreEntry](#) objects may or may not be available. An entry is unavailable if it has a private content that is not present. The private content might exist on external hardware. To determine if an entry is available, call [isAvailable\(\)](#). To ensure an entry is available before performing a private key operation, call [ensureAvailable](#). For example:

```
if(entry.ensureAvailable())
{
    entry.keyBundle().privateKey().signMessage(...);
    ...
}
```

[ensureAvailable\(\)](#) blocks and may cause hardware access, but if it completes successfully then you may use the entry's private content. It also means, in the case of a Smart Card token, that it is probably inserted.

To watch this entry asynchronously, you would do:

```
KeyStoreEntryWatcher *watcher = new KeyStoreEntryWatcher(entry);
connect(watcher, SIGNAL(available()), SLOT(entry_available()));
...
void entry_available()
{
    // entry now available
    watcher->entry().keyBundle().privateKey().signMessage(...);
}
```

Unlike private content, public content is always usable even if the entry is not available. Serialized entry data contains all of the metadata necessary to reconstruct the public content.

Now, even though an entry may be available, it does not mean you have access to use it for operations. For example, even though a [KeyBundle](#) entry offered by a Smart Card may be available, as soon as you try to use the [PrivateKey](#) object for a signing operation, a PIN might be asked for. You can call [ensureAccess\(\)](#) if you want to synchronously provide the PIN early on:

```
if(entry.ensureAccess())
{
    // do private key stuff
    ...
}
```

Note that you don't have to call [ensureAvailable\(\)](#) before [ensureAccess\(\)](#). Calling the latter is enough to imply both.

After an application is configured to use a particular key, it is expected that its usual running procedure will be:

1) Construct [KeyStoreEntry](#) from the serialized data. 2) If the content object is not available, wait for it (with either [ensureAvailable\(\)](#) or [KeyStoreEntryWatcher](#)). 3) Pass the content object(s) to a high level operation like [TLS](#).

In this case, any PIN prompting and private key operations would be caused/handled from the [TLS](#) object. Omit step 2 and the private key operations might cause token prompting.

Examples:

[eventhandlerdemo.cpp](#).

10.40.2 Member Enumeration Documentation

10.40.2.1 enum [QCA::KeyStoreEntry::Type](#)

The type of entry in the [KeyStore](#).

10.40.3 Constructor & Destructor Documentation

10.40.3.1 [QCA::KeyStoreEntry::KeyStoreEntry \(\)](#)

Create an empty [KeyStoreEntry](#).

10.40.3.2 [QCA::KeyStoreEntry::KeyStoreEntry \(const QString & *serialized*\)](#)

Create a passive [KeyStoreEntry](#) based on a serialized string.

See also:

[fromString](#)

10.40.3.3 [QCA::KeyStoreEntry::KeyStoreEntry \(const \[KeyStoreEntry\]\(#\) & *from*\)](#)

Standard copy constructor.

Parameters:

from the source entry

10.40.4 Member Function Documentation

10.40.4.1 [KeyStoreEntry](#)& QCA::KeyStoreEntry::operator= (const [KeyStoreEntry](#) & *from*)

Standard assignment operator.

Parameters:

from the source entry

10.40.4.2 bool QCA::KeyStoreEntry::isNull () const

Test if this key is empty (null).

10.40.4.3 bool QCA::KeyStoreEntry::isAvailable () const

Test if the key is available for use.

A key is considered available if the key's private content is present.

See also:

[ensureAvailable](#)
[isAccessible](#)

10.40.4.4 bool QCA::KeyStoreEntry::isAccessible () const

Test if the key is currently accessible.

This means that the private key part can be used at this time. For a smartcard, this means that all required operations (e.g. login / PIN entry) are completed.

If [isAccessible\(\)](#) is true, then the key is necessarily available (i.e. [isAvailable\(\)](#) is also true).

See also:

[ensureAccessible](#)
[isAvailable](#)

10.40.4.5 [Type](#) QCA::KeyStoreEntry::type () const

Determine the type of key stored in this object.

Reimplemented from [QCA::Algorithm](#).

10.40.4.6 QString QCA::KeyStoreEntry::name () const

The name associated with the key stored in this object.

10.40.4.7 QString QCA::KeyStoreEntry::id () const

The ID associated with the key stored in this object.

10.40.4.8 QString QCA::KeyStoreEntry::storeName () const

The name of the [KeyStore](#) for this key object.

10.40.4.9 QString QCA::KeyStoreEntry::storeId () const

The id of the [KeyStore](#) for this key object.

See also:

[KeyStore::id\(\)](#)

10.40.4.10 QString QCA::KeyStoreEntry::toString () const

Serialize into a string for use as a passive entry.

**10.40.4.11 static [KeyStoreEntry](#) QCA::KeyStoreEntry::fromString (const QString & *serialized*)
[static]**

Load a passive entry by using a serialized string as input.

Returns:

the newly created [KeyStoreEntry](#)

10.40.4.12 [KeyBundle](#) QCA::KeyStoreEntry::keyBundle () const

If a [KeyBundle](#) is stored in this object, return that bundle.

10.40.4.13 [Certificate](#) QCA::KeyStoreEntry::certificate () const

If a [Certificate](#) is stored in this object, return that certificate.

10.40.4.14 [CRL](#) QCA::KeyStoreEntry::crl () const

If a [CRL](#) is stored in this object, return the value of the [CRL](#).

10.40.4.15 [PGPKey](#) QCA::KeyStoreEntry::pgpSecretKey () const

If the key stored in this object is a private PGP key, return the contents of that key.

10.40.4.16 [PGPKey](#) QCA::KeyStoreEntry::pgpPublicKey () const

If the key stored in this object is either an public or private PGP key, extract the public key part of that PGP key.

10.40.4.17 bool QCA::KeyStoreEntry::ensureAvailable ()

Returns true if the entry is available, otherwise false.

Available means that any private content for this entry is present and ready for use. In the case of a smart card, this will ensure the card is inserted, and may invoke a token prompt.

Calling this function on an already available entry may cause the entry to be refreshed.

See also:

[isAvailable](#)
[ensureAccess](#)

Note:

This function is blocking.

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.40.4.18 bool QCA::KeyStoreEntry::ensureAccess ()

Like ensureAvailable, but will also ensure that the PIN is provided if needed.

See also:

[isAccessible](#)
[ensureAvailable](#)

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

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

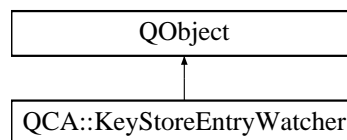
- [qca_keystore.h](#)

10.41 QCA::KeyStoreEntryWatcher Class Reference

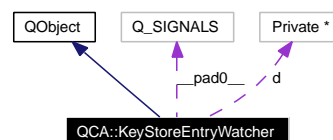
Class to monitor the availability of a [KeyStoreEntry](#).

```
#include <qca_keystore.h>
```

Inheritance diagram for QCA::KeyStoreEntryWatcher::



Collaboration diagram for QCA::KeyStoreEntryWatcher:



Public Member Functions

- [KeyStoreEntryWatcher](#) (const [KeyStoreEntry](#) &e, [QObject](#) *parent=0)
- [KeyStoreEntry](#) entry () const
- void [unavailable](#) ()

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void available()

Friends

- class [Private](#)

10.41.1 Detailed Description

Class to monitor the availability of a [KeyStoreEntry](#).

Some [KeyStore](#) types have the concept of an entry that can be available only part of the time (for example, a smart card that can be removed). This class allows you to identify when a [KeyStoreEntry](#) becomes available / unavailable.

Note:

You can also monitor availability of a whole [KeyStore](#), using [KeyStoreManager::keyStoreAvailable\(\)](#) signal, and the [KeyStore::unavailable\(\)](#) signal.

See also:

[KeyStore](#) for more discussion on availability of keys and related objects.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 `QCA::KeyStoreEntryWatcher::KeyStoreEntryWatcher (const KeyStoreEntry & e, QObject *parent = 0) [explicit]`

Standard constructor.

This creates an object that monitors the specified [KeyStore](#) entry, emitting `available()` and `unavailable()` as the entry becomes available and unavailable respectively.

Parameters:

- e* the [KeyStoreEntry](#) to monitor
- parent* the parent object for this object

10.41.3 Member Function Documentation

10.41.3.1 [KeyStoreEntry](#) `QCA::KeyStoreEntryWatcher::entry () const`

The [KeyStoreEntry](#) that is being monitored.

10.41.3.2 `void QCA::KeyStoreEntryWatcher::unavailable ()`

This signal is emitted when the entry that is being monitored becomes unavailable.

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

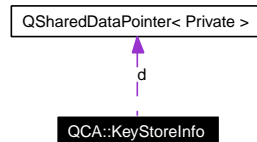
- [qca_keystore.h](#)

10.42 QCA::KeyStoreInfo Class Reference

Key store information, outside of a [KeyStore](#) object.

```
#include <qca_keystore.h>
```

Collaboration diagram for QCA::KeyStoreInfo:



Public Member Functions

- [KeyStoreInfo](#) ()
- [KeyStoreInfo](#) ([KeyStore::Type](#) type, const [QString](#) &id, const [QString](#) &name)
- [KeyStoreInfo](#) (const [KeyStoreInfo](#) &from)
- [KeyStoreInfo](#) & operator= (const [KeyStoreInfo](#) &from)
- bool [isNull](#) () const
- [KeyStore::Type](#) type () const
- [QString](#) id () const
- [QString](#) name () const

10.42.1 Detailed Description

Key store information, outside of a [KeyStore](#) object.

This class is used in conjunction with the [Event](#) class, and related classes such as [PasswordAsker](#) and [TokenAsker](#), to describe the key store source of the [Event](#).

Each [KeyStoreInfo](#) represents a single [KeyStore](#), and describes the type of store (e.g. smartcard or PGP keyring - see [KeyStore::Type](#)), and a couple of names. The [id\(\)](#) of a [KeyStore](#) is used to reference it, and is typically of the form "qca-mystorename". The [name\(\)](#) of a [KeyStore](#) is used to describe it (i.e. this is the "pretty" name to show the user), and is typically of the form "My Store Name".

Examples:

[eventhandlerdemo.cpp](#).

10.42.2 Constructor & Destructor Documentation

10.42.2.1 QCA::KeyStoreInfo::KeyStoreInfo ()

Constructor.

Note:

This form of constructor for [KeyStoreInfo](#) produces an object that does not describe any [KeyStore](#), and [isNull\(\)](#) will return true.

10.42.2.2 QCA::KeyStoreInfo::KeyStoreInfo ([KeyStore::Type](#) *type*, const QString & *id*, const QString & *name*)

Standard constructor.

This builds a [KeyStoreInfo](#) object that describes a [KeyStore](#).

Parameters:

type the type of [KeyStore](#)

id the identification of the [KeyStore](#)

name the descriptive name of the [KeyStore](#)

10.42.2.3 QCA::KeyStoreInfo::KeyStoreInfo (const [KeyStoreInfo](#) & *from*)

Copy constructor.

Parameters:

from the [KeyStoreInfo](#) to copy from

10.42.3 Member Function Documentation

10.42.3.1 [KeyStoreInfo](#)& QCA::KeyStoreInfo::operator= (const [KeyStoreInfo](#) & *from*)

Assignment operator.

Parameters:

from the [KeyStoreInfo](#) to copy from

10.42.3.2 bool QCA::KeyStoreInfo::isNull () const

Test if this object is valid.

Returns:

true if the object is not valid

10.42.3.3 [KeyStore::Type](#) QCA::KeyStoreInfo::type () const

The Type of [KeyStore](#) that this [KeyStoreInfo](#) object describes.

10.42.3.4 QString QCA::KeyStoreInfo::id () const

The unique identification of the [KeyStore](#) that this [KeyStoreInfo](#) object describes.

10.42.3.5 QString QCA::KeyStoreInfo::name () const

The descriptive name of the [KeyStore](#) that this [KeyStoreInfo](#) object describes.

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

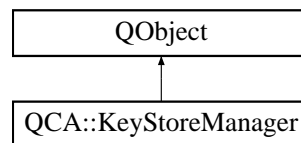
- [qca_keystore.h](#)

10.43 QCA::KeyStoreManager Class Reference

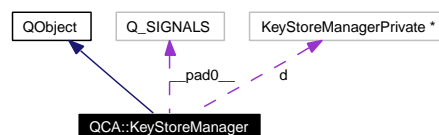
Access keystores, and monitor keystores for changes.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::KeyStoreManager:::



Collaboration diagram for QCA::KeyStoreManager:



Public Member Functions

- [KeyStoreManager](#) ([QObject](#) *parent=0)
- bool [isBusy](#) () const
- void [waitForBusyFinished](#) ()
- [QStringList](#) [keyStores](#) () const
- void [sync](#) ()
- void [busyFinished](#) ()
- void [keyStoreAvailable](#) (const [QString](#) &id)

Static Public Member Functions

- static void [start](#) ()
- static void [start](#) (const [QString](#) &provider)
- static [QString](#) [diagnosticText](#) ()
- static void [clearDiagnosticText](#) ()

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void busyStarted()

Friends

- class [KeyStoreManagerPrivate](#)
- class [Global](#)
- class [KeyStorePrivate](#)

10.43.1 Detailed Description

Access keystores, and monitor keystores for changes.

Before you can access a [KeyStore](#), you must create a [KeyStoreManager](#). You then need to [start\(\)](#) the [KeyStoreManager](#), and either wait for the [busyFinished\(\)](#) signal, or block using [waitForBusyFinished\(\)](#).

If you know the [KeyStoreEntry](#) that you need, you can use [KeyStore](#) passively, as described in the [KeyStoreEntry](#) documentation.

10.43.2 Constructor & Destructor Documentation

10.43.2.1 `QCA::KeyStoreManager::KeyStoreManager (QObject * parent = 0)`

Create a new [KeyStoreManager](#).

Parameters:

parent the parent for this object

10.43.3 Member Function Documentation

10.43.3.1 `static void QCA::KeyStoreManager::start ()` [static]

Initialize all key store providers.

10.43.3.2 `static void QCA::KeyStoreManager::start (const QString & provider)` [static]

Initialize a specific key store provider.

10.43.3.3 `bool QCA::KeyStoreManager::isBusy () const`

Indicates if the manager is busy looking for key stores.

10.43.3.4 `void QCA::KeyStoreManager::waitForBusyFinished ()`

Blocks until the manager is done looking for key stores.

10.43.3.5 `QStringList QCA::KeyStoreManager::keyStores () const`

A list of all the key stores.

10.43.3.6 `static QString QCA::KeyStoreManager::diagnosticText ()` [static]

The diagnostic result of key store operations, such as warnings and errors.

10.43.3.7 `static void QCA::KeyStoreManager::clearDiagnosticText ()` [static]

Clears the diagnostic result log.

10.43.3.8 void QCA::KeyStoreManager::sync ()

If you are not using the eventloop, call this to update the object state to the present.

10.43.3.9 void QCA::KeyStoreManager::busyFinished ()

emitted when the manager has finished looking for key stores

10.43.3.10 void QCA::KeyStoreManager::keyStoreAvailable (const QString & *id*)

emitted when a new key store becomes available

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

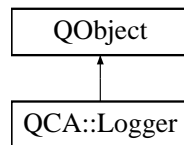
- [qca_keystore.h](#)

10.44 QCA::Logger Class Reference

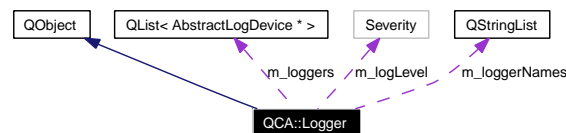
A simple logging system.

```
#include <qca_support.h>
```

Inheritance diagram for QCA::Logger::



Collaboration diagram for QCA::Logger:



Public Types

- enum [Severity](#) {
 [Quiet](#) = 0, [Emergency](#) = 1, [Alert](#) = 2, [Critical](#) = 3,
 [Error](#) = 4, [Warning](#) = 5, [Notice](#) = 6, [Information](#) = 7,
 [Debug](#) = 8 }

Public Member Functions

- [Logger::Severity level](#) () const
- void [setLevel](#) ([Logger::Severity](#) level)
- void [logTextMessage](#) (const **QString** &message, [Severity](#)=[Information](#))
- void [logBinaryMessage](#) (const **QByteArray** &blob, [Severity](#)=[Information](#))
- void [registerLogDevice](#) ([AbstractLogDevice](#) *logger)
- void [unregisterLogDevice](#) (const **QString** &loggerName)
- **QStringList** [currentLogDevices](#) () const

Friends

- class **Global**

10.44.1 Detailed Description

A simple logging system.

This class provides a simple but flexible approach to logging information that may be used for debugging or system operation diagnostics.

There is a single `Logger` for each application that uses QCA. You do not need to create this `Logger` yourself - QCA automatically creates it on startup. You can get access to the `Logger` using the global `QCA::logger()` method.

By default the `Logger` just accepts all messages (binary and text). If you want to get access to those messages, you need to subclass `AbstractLogDevice`, and register your subclass (using `registerLogDevice()`). You can then take whatever action is appropriate (e.g. show to the user using the GUI, log to a file or send to standard error).

10.44.2 Member Enumeration Documentation

10.44.2.1 enum `QCA::Logger::Severity`

The severity of the message.

This information may be used by the log device to determine what the appropriate action is.

Enumerator:

Quiet Quiet: turn of logging.

Emergency Emergency: system is unusable.

Alert Alert: action must be taken immediately.

Critical Critical: critical conditions.

Error Error: error conditions.

Warning Warning: warning conditions.

Notice Notice: normal but significant condition.

Information Informational: informational messages.

Debug Debug: debug-level messages.

10.44.3 Member Function Documentation

10.44.3.1 `Logger::Severity` `QCA::Logger::level () const` [inline]

Get the current logging level.

Returns:

Current level

10.44.3.2 `void QCA::Logger::setLevel (Logger::Severity level)`

Set the current logging level.

Parameters:

level new logging level

Only severities less or equal than the log level one will be logged

10.44.3.3 void QCA::Logger::logTextMessage (const QString & *message*, Severity = Information)

Log a message to all available log devices.

Parameters:

message the text to log

10.44.3.4 void QCA::Logger::logBinaryMessage (const QByteArray & *blob*, Severity = Information)

Log a binary blob to all available log devices.

Parameters:

blob the information to log

Note:

how this is handled is quite logger specific. For example, it might be logged as a binary, or it might be encoded in some way

10.44.3.5 void QCA::Logger::registerLogDevice (AbstractLogDevice * *logger*)

Add an [AbstractLogDevice](#) subclass to the existing list of loggers.

Parameters:

logger the LogDevice to add

10.44.3.6 void QCA::Logger::unregisterLogDevice (const QString & *loggerName*)

Remove an [AbstractLogDevice](#) subclass from the existing list of loggers.

Parameters:

loggerName the name of the LogDevice to remove

Note:

If there are several log devices with the same name, all will be removed.

10.44.3.7 QStringList QCA::Logger::currentLogDevices () const

Get a list of the names of all registered log devices.

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

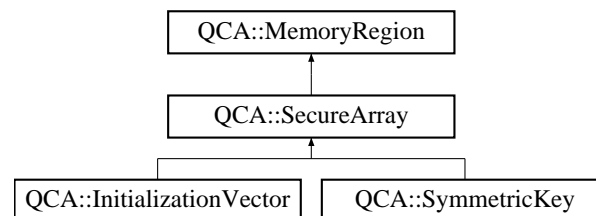
- [qca_support.h](#)

10.45 QCA::MemoryRegion Class Reference

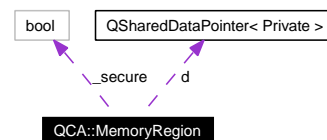
Array of bytes that may be optionally secured.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::MemoryRegion::



Collaboration diagram for QCA::MemoryRegion:



Public Member Functions

- [MemoryRegion](#) (const char *str)
- [MemoryRegion](#) (const **QByteArray** &from)
- [MemoryRegion](#) (const [MemoryRegion](#) &from)
- [MemoryRegion](#) & operator= (const [MemoryRegion](#) &from)
- [MemoryRegion](#) & operator= (const **QByteArray** &from)
- bool [isNull](#) () const
- bool [isSecure](#) () const
- **QByteArray** [toByteArray](#) () const
- bool [isEmpty](#) () const
- int [size](#) () const
- const char * [data](#) () const
- const char * [constData](#) () const
- const char & [at](#) (int index) const

Protected Member Functions

- [MemoryRegion](#) (bool secure)
- [MemoryRegion](#) (int size, bool secure)
- [MemoryRegion](#) (const **QByteArray** &from, bool secure)
- char * [data](#) ()
- char & [at](#) (int index)
- bool [resize](#) (int size)
- void [set](#) (const **QByteArray** &from, bool secure)
- void [setSecure](#) (bool secure)

10.45.1 Detailed Description

Array of bytes that may be optionally secured.

This class is mostly unusable on its own. Either use it as a [SecureArray](#) subclass or call [toByteArray\(\)](#) to convert to **QByteArray**.

Note that this class is implicitly shared (that is, copy on write).

Examples:

[aes-cmac.cpp](#).

10.45.2 Constructor & Destructor Documentation

10.45.2.1 QCA::MemoryRegion::MemoryRegion (const char * *str*)

Constructs a new Memory Region from a null terminated character array.

Parameters:

str pointer to the array of data to copy

10.45.2.2 QCA::MemoryRegion::MemoryRegion (const QByteArray & *from*)

Constructs a new [MemoryRegion](#) from the data in a byte array.

10.45.2.3 QCA::MemoryRegion::MemoryRegion (const [MemoryRegion](#) & *from*)

Standard copy constructor.

10.45.2.4 QCA::MemoryRegion::MemoryRegion (bool *secure*) [protected]

Create a memory region, optionally using secure storage.

Parameters:

secure if this is true, the memory region will use secure storage.

Note:

This will create a memory region without any content (i.e. both [isNull\(\)](#) and [isEmpty\(\)](#) will return true.

10.45.2.5 QCA::MemoryRegion::MemoryRegion (int *size*, bool *secure*) [protected]

Create a memory region, optionally using secure storage.

Parameters:

size the number of bytes in the memory region.

secure if this is true, the memory region will use secure storage.

10.45.2.6 QCA::MemoryRegion::MemoryRegion (const QByteArray &from, bool secure) [protected]

Create a memory region, optionally using secure storage.

This constructor variant allows you to initialize the memory region from an existing array.

Parameters:

from the byte array to copy from.

secure if this is true, the memory region will use secure storage.

10.45.3 Member Function Documentation

10.45.3.1 MemoryRegion& QCA::MemoryRegion::operator= (const MemoryRegion &from)

Standard assignment operator.

10.45.3.2 MemoryRegion& QCA::MemoryRegion::operator= (const QByteArray &from)

Standard assignment operator.

Reimplemented in [QCA::SecureArray](#).

10.45.3.3 bool QCA::MemoryRegion::isNull () const

Test if the [MemoryRegion](#) is null (i.e.

was created as a null array, and hasn't been resized).

This is probably not what you are trying to do. If you are trying to determine whether there are any bytes in the array, use [isEmpty\(\)](#) instead.

10.45.3.4 bool QCA::MemoryRegion::isSecure () const

Test if the [MemoryRegion](#) is using secure memory, or not.

In this context, memory is secure if it will not be paged out to disk.

Returns:

true if the memory region is secure

10.45.3.5 QByteArray QCA::MemoryRegion::toByteArray () const

Convert this memory region to a byte array.

Note:

For secure data, this will make it insecure

See also:

[data\(\)](#) and [constData\(\)](#) for other ways to convert to an "accessible" format.

Reimplemented in [QCA::SecureArray](#).

10.45.3.6 bool QCA::MemoryRegion::isEmpty () const

Returns true if the size of the memory region is zero.

Reimplemented in [QCA::SecureArray](#).

10.45.3.7 int QCA::MemoryRegion::size () const

Returns the number of bytes in the memory region.

Reimplemented in [QCA::SecureArray](#).

10.45.3.8 const char* QCA::MemoryRegion::data () const

Convert the contents of the memory region to a C-compatible character array.

This consists of [size\(\)](#) bytes, followed by a null terminator.

See also:

[toByteArray](#) for an alternative approach.

[constData](#), which is equivalent to this method, but avoids the possibility that the compiler picks the wrong version.

Reimplemented in [QCA::SecureArray](#).

10.45.3.9 const char* QCA::MemoryRegion::constData () const

Convert the contents of the memory region to a C-compatible character array.

This consists of [size\(\)](#) bytes, followed by a null terminator.

See also:

[toByteArray](#) for an alternative approach.

[data](#) which is equivalent to this method

Reimplemented in [QCA::SecureArray](#).

10.45.3.10 const char& QCA::MemoryRegion::at (int *index*) const

Obtain the value of the memory location at the specified position.

Parameters:

index the offset into the memory region.

Note:

The contents of a memory region are between 0 and [size\(\)](#)-1. The content at position [size\(\)](#) is always a null terminator.

Reimplemented in [QCA::SecureArray](#).

10.45.3.11 `char* QCA::MemoryRegion::data ()` [protected]

Convert the contents of the memory region to a C-compatible character array.

This consists of `size()` bytes, followed by a null terminator.

Reimplemented in [QCA::SecureArray](#).

10.45.3.12 `char& QCA::MemoryRegion::at (int index)` [protected]

Obtain the value of the memory location at the specified position.

Parameters:

index the offset into the memory region.

Note:

The contents of a memory region are between 0 and `size()-1`. The content at position `size()` is always a null terminator.

Reimplemented in [QCA::SecureArray](#).

10.45.3.13 `bool QCA::MemoryRegion::resize (int size)` [protected]

Resize the memory region to the specified size.

Parameters:

size the new size of the region.

Reimplemented in [QCA::SecureArray](#).

10.45.3.14 `void QCA::MemoryRegion::set (const QByteArray &from, bool secure)` [protected]

Modify the memory region to match a specified byte array.

This resizes the memory region as required to match the byte array size.

Parameters:

from the byte array to copy from.

secure if this is true, the memory region will use secure storage.

10.45.3.15 `void QCA::MemoryRegion::setSecure (bool secure)` [protected]

Convert the memory region to use the specified memory type.

This may involve copying data from secure to insecure storage, or from insecure to secure storage.

Parameters:

secure if true, use secure memory; otherwise use insecure memory.

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

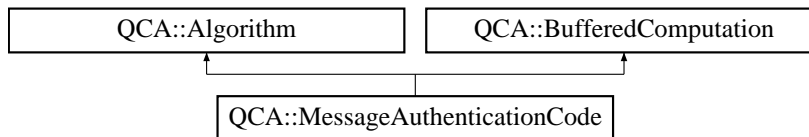
- [qca_tools.h](#)

10.46 QCA::MessageAuthenticationCode Class Reference

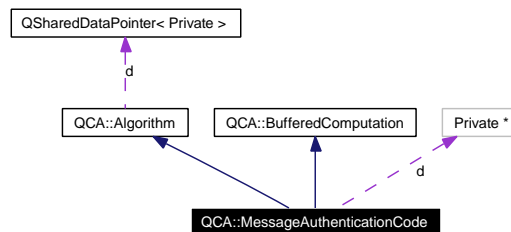
General class for message authentication code (MAC) algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::MessageAuthenticationCode::



Collaboration diagram for QCA::MessageAuthenticationCode:



Public Member Functions

- `MessageAuthenticationCode` (const `QString` &type, const `SymmetricKey` &key, const `QString` &provider=`QString`())
- `MessageAuthenticationCode` (const `MessageAuthenticationCode` &from)
- `MessageAuthenticationCode` & operator= (const `MessageAuthenticationCode` &from)
- `QString` type () const
- `KeyLength` keyLength () const
- bool `validKeyLength` (int n) const
- virtual void `clear` ()
- virtual void `update` (const `MemoryRegion` &array)
- virtual `MemoryRegion` `final` ()
- void `setup` (const `SymmetricKey` &key)

10.46.1 Detailed Description

General class for message authentication code (MAC) algorithms.

`MessageAuthenticationCode` is a class for accessing the various message authentication code algorithms within QCA. HMAC using SHA1 ("hmac(sha1)") or HMAC using SHA256 ("hmac(sha256)") is recommended for new applications.

Note that if your application is potentially susceptible to "replay attacks" where the message is sent more than once, you should include a counter in the message that is covered by the MAC, and check that the counter is always incremented every time you receive a message and MAC.

For more information on HMAC, see H. Krawczyk et al. RFC2104 "HMAC: Keyed-Hashing for Message Authentication"

Examples:

[mactest.cpp](#).

10.46.2 Constructor & Destructor Documentation

10.46.2.1 QCA::MessageAuthenticationCode::MessageAuthenticationCode (const QString & *type*, const [SymmetricKey](#) & *key*, const QString & *provider* = QString ())

Standard constructor.

Parameters:

type the name of the MAC (and algorithm, if applicable) to use

key the shared key

provider the provider to use, if a particular provider is required

10.46.2.2 QCA::MessageAuthenticationCode::MessageAuthenticationCode (const [MessageAuthenticationCode](#) & *from*)

Standard copy constructor.

10.46.3 Member Function Documentation

10.46.3.1 [MessageAuthenticationCode&](#) QCA::MessageAuthenticationCode::operator= (const [MessageAuthenticationCode](#) & *from*)

Assignment operator.

Copies the state (including key) from one [MessageAuthenticationCode](#) to another

10.46.3.2 QString QCA::MessageAuthenticationCode::type () const

Return the MAC type.

Reimplemented from [QCA::Algorithm](#).

10.46.3.3 [KeyLength](#) QCA::MessageAuthenticationCode::keyLength () const

Return acceptable key lengths.

10.46.3.4 bool QCA::MessageAuthenticationCode::validKeyLength (int *n*) const

Test if a key length is valid for the MAC algorithm.

Parameters:

n the key length in bytes

Returns:

true if the key would be valid for the current algorithm

10.46.3.5 virtual void QCA::MessageAuthenticationCode::clear () [virtual]

Reset a [MessageAuthenticationCode](#), dumping all previous parts of the message.

This method clears (or resets) the algorithm, effectively undoing any previous [update\(\)](#) calls. You should use this call if you are re-using a MessageAuthenticationCode sub-class object to calculate additional MACs. Note that if the key doesn't need to be changed, you don't need to call [setup\(\)](#) again, since the key can just be reused.

Implements [QCA::BufferedComputation](#).

10.46.3.6 virtual void QCA::MessageAuthenticationCode::update (const [MemoryRegion](#) & array) [virtual]

Update the MAC, adding more of the message contents to the digest.

The whole message needs to be added using this method before you call [final\(\)](#).

Parameters:

array the message contents

Implements [QCA::BufferedComputation](#).

Examples:

[mactest.cpp](#).

10.46.3.7 virtual [MemoryRegion](#) QCA::MessageAuthenticationCode::final () [virtual]

Finalises input and returns the MAC result.

After calling [update\(\)](#) with the required data, the hash results are finalised and produced.

Note that it is not possible to add further data (with [update\(\)](#)) after calling [final\(\)](#). If you want to reuse the MessageAuthenticationCode object, you should call [clear\(\)](#) and start to [update\(\)](#) again.

Implements [QCA::BufferedComputation](#).

Examples:

[mactest.cpp](#).

10.46.3.8 void QCA::MessageAuthenticationCode::setup (const [SymmetricKey](#) & key)

Initialise the MAC algorithm.

Parameters:

key the key to use for the algorithm

Examples:

[mactest.cpp](#).

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

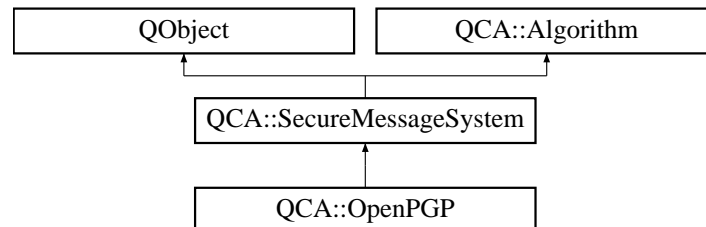
- [qca_basic.h](#)

10.47 QCA::OpenPGP Class Reference

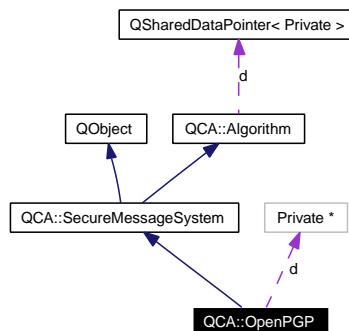
Pretty Good Privacy messaging system.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::OpenPGP::



Collaboration diagram for QCA::OpenPGP:



Public Member Functions

- [OpenPGP](#) (`QObject *parent=0`, `const QString &provider=QString()`)

10.47.1 Detailed Description

Pretty Good Privacy messaging system.

See also:

[SecureMessage](#)
[SecureMessageKey](#)

10.47.2 Constructor & Destructor Documentation

10.47.2.1 QCA::OpenPGP::OpenPGP (QObject *parent = 0, const QString &provider = QString()) [explicit]

Standard constructor.

Parameters:

parent the parent object for this object

provider the provider to use, if a specific provider is required

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

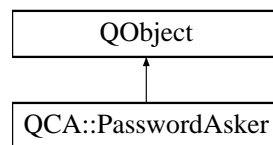
- [qca_securemessage.h](#)

10.48 QCA::PasswordAsker Class Reference

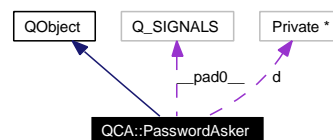
User password / passphrase / PIN handler.

```
#include <qca_core.h>
```

Inheritance diagram for QCA::PasswordAsker::



Collaboration diagram for QCA::PasswordAsker:



Public Member Functions

- [PasswordAsker](#) ([QObject](#) *parent=0)
- void [ask](#) ([Event::PasswordStyle](#) pstyle, const [KeyStoreInfo](#) &keyStoreInfo, const [KeyStoreEntry](#) &keyStoreEntry, void *ptr)
- void [ask](#) ([Event::PasswordStyle](#) pstyle, const [QString](#) &fileName, void *ptr)
- void [cancel](#) ()
- void [waitForResponse](#) ()
- bool [accepted](#) () const
- [SecureArray](#) [password](#) () const

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void responseReady()

Friends

- class [Private](#)

10.48.1 Detailed Description

User password / passphrase / PIN handler.

This class is used to obtain a password from a user.

Examples:

[eventhandlerdemo.cpp](#).

10.48.2 Constructor & Destructor Documentation

10.48.2.1 QCA::PasswordAsker::PasswordAsker (QObject * *parent* = 0)

Construct a new asker.

Parameters:

parent the parent object for this **QObject**

10.48.3 Member Function Documentation

10.48.3.1 void QCA::PasswordAsker::ask (Event::PasswordStyle *pstyle*, const **KeyStoreInfo** & *keyStoreInfo*, const **KeyStoreEntry** & *keyStoreEntry*, void * *ptr*)

queue a password / passphrase request associated with a key store

Parameters:

pstyle the type of information required (e.g. PIN, passphrase or password)

keyStoreInfo info of the key store that the information is required for

keyStoreEntry the item in the key store that the information is required for (if applicable)

ptr opaque data

Examples:

[eventhandlerdemo.cpp](#).

10.48.3.2 void QCA::PasswordAsker::ask (Event::PasswordStyle *pstyle*, const QString & *fileName*, void * *ptr*)

queue a password / passphrase request associated with a file

Parameters:

pstyle the type of information required (e.g. PIN, passphrase or password)

fileName the name of the file that the information is required for

ptr opaque data

10.48.3.3 void QCA::PasswordAsker::cancel ()

Cancel the pending password / passphrase request.

10.48.3.4 void QCA::PasswordAsker::waitForResponse ()

Block until the password / passphrase request is completed.

You can use the responseReady signal instead of blocking, if appropriate.

Examples:

[eventhandlerdemo.cpp](#).

10.48.3.5 `bool QCA::PasswordAsker::accepted () const`

Determine whether the password / passphrase was accepted or not.

In this context, returning true is indicative of the user clicking "Ok" or equivalent; and returning false indicates that either the user clicked "Cancel" or equivalent, or that the [cancel\(\)](#) function was called, or that the request is still pending.

10.48.3.6 `SecureArray QCA::PasswordAsker::password () const`

The password / passphrase / PIN provided by the user in response to the asker request.

This may be empty.

Examples:

[eventhandlerdemo.cpp](#).

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

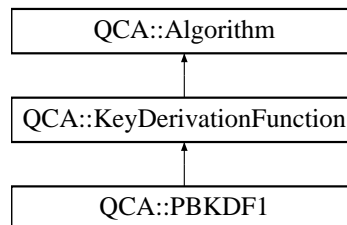
- [qca_core.h](#)

10.49 QCA::PBKDF1 Class Reference

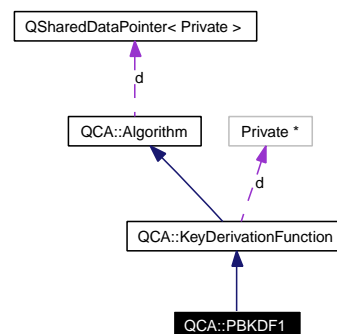
Password based key derivation function version 1.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PBKDF1::



Collaboration diagram for QCA::PBKDF1:



Public Member Functions

- [PBKDF1](#) (const **QString** &algorithm="sha1", const **QString** &provider=**QString**())

10.49.1 Detailed Description

Password based key derivation function version 1.

This class implements Password Based Key Derivation Function version 1, as specified in RFC2898, and also in PKCS#5.

10.49.2 Constructor & Destructor Documentation

10.49.2.1 QCA::PBKDF1::PBKDF1 (const **QString** & *algorithm* = "sha1", const **QString** & *provider* = **QString**()) [inline, explicit]

Standard constructor.

Parameters:

algorithm the name of the hashing algorithm to use

provider the name of the provider to use, if available

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

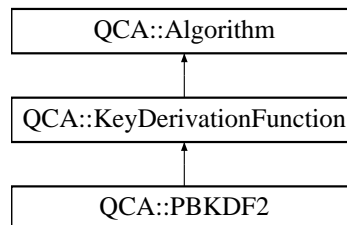
- [qca_basic.h](#)

10.50 QCA::PBKDF2 Class Reference

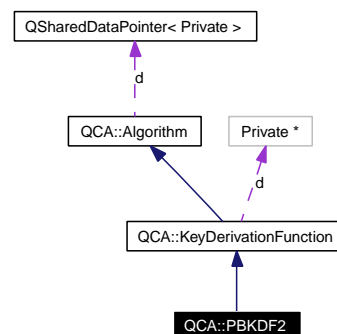
Password based key derivation function version 2.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PBKDF2::



Collaboration diagram for QCA::PBKDF2:



Public Member Functions

- [PBKDF2](#) (const **QString** &algorithm="sha1", const **QString** &provider=**QString**())

10.50.1 Detailed Description

Password based key derivation function version 2.

This class implements Password Based Key Derivation Function version 2, as specified in RFC2898, and also in PKCS#5.

10.50.2 Constructor & Destructor Documentation

10.50.2.1 QCA::PBKDF2::PBKDF2 (const **QString** & *algorithm* = "sha1", const **QString** & *provider* = **QString**()) [inline, explicit]

Standard constructor.

Parameters:

algorithm the name of the hashing algorithm to use

provider the name of the provider to use, if available

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

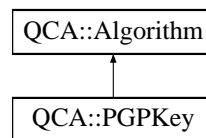
- [qca_basic.h](#)

10.51 QCA::PGPKey Class Reference

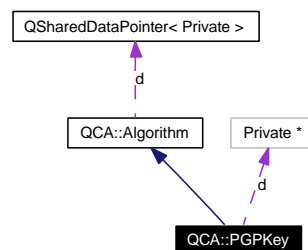
Pretty Good Privacy key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PGPKey::



Collaboration diagram for QCA::PGPKey:



Public Member Functions

- [PGPKey \(\)](#)
- [PGPKey \(const QString &fileName\)](#)
- [PGPKey \(const PGPKey &from\)](#)
- [PGPKey & operator= \(const PGPKey &from\)](#)
- [bool isNull \(\) const](#)
- [QString keyId \(\) const](#)
- [QString primaryUserId \(\) const](#)
- [QStringList userIds \(\) const](#)
- [bool isSecret \(\) const](#)
- [QDateTime creationDate \(\) const](#)
- [QDateTime expirationDate \(\) const](#)
- [QString fingerprint \(\) const](#)
- [bool inKeyring \(\) const](#)
- [bool isTrusted \(\) const](#)
- [QByteArray toArray \(\) const](#)
- [QString toString \(\) const](#)
- [bool toFile \(const QString &fileName\) const](#)

Static Public Member Functions

- [static PGPKey fromArray \(const QByteArray &a, ConvertResult *result=0, const QString &provider=QString\(\)\)](#)

- static [PGPKey fromString](#) (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString\(\)](#))
- static [PGPKey fromFile](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString\(\)](#))

10.51.1 Detailed Description

Pretty Good Privacy key.

This holds either a reference to an item in a real PGP keyring, or a standalone item created using the from*() functions.

Note that with the latter method, the key is of no use besides being informational. The key must be in a keyring (that is, [inKeyring\(\)](#) == true) to actually do crypto with it.

10.51.2 Constructor & Destructor Documentation

10.51.2.1 QCA::PGPKey::PGPKey ()

Create an empty PGP key.

10.51.2.2 QCA::PGPKey::PGPKey (const [QString](#) & *fileName*)

Create a PGP key from an encoded file.

See also:

[fromFile](#)
[toFile](#)

10.51.2.3 QCA::PGPKey::PGPKey (const [PGPKey](#) & *from*)

Standard copy constructor.

Parameters:

from the [PGPKey](#) to use as the source

10.51.3 Member Function Documentation

10.51.3.1 [PGPKey&](#) QCA::PGPKey::operator= (const [PGPKey](#) & *from*)

Standard assignment operator.

Parameters:

from the [PGPKey](#) to use as the source

10.51.3.2 bool QCA::PGPKey::isNull () const

Test if the PGP key is empty (null).

Returns:

true if the PGP key is null

10.51.3.3 QString QCA::PGPKey::keyId () const

The Key identification for the PGP key.

10.51.3.4 QString QCA::PGPKey::primaryUserId () const

The primary user identification for the key.

10.51.3.5 QStringList QCA::PGPKey::userIds () const

The list of all user identifications associated with the key.

10.51.3.6 bool QCA::PGPKey::isSecret () const

Test if the PGP key is the secret key.

Returns:

true if the PGP key is the secret key

10.51.3.7 QDateTime QCA::PGPKey::creationDate () const

The creation date for the key.

10.51.3.8 QDateTime QCA::PGPKey::expirationDate () const

The expiration date for the key.

10.51.3.9 QString QCA::PGPKey::fingerprint () const

The key fingerprint.

This will return the PGP fingerprint as a string. It comprises 16 hex digits, without spaces.

10.51.3.10 bool QCA::PGPKey::inKeyring () const

Test if this key is in a keyring.

Returns:

true if the key is in a keyring

Note:

keys that are not in a keyring cannot be used for encryption, decryption, signing or verification

10.51.3.11 bool QCA::PGPKey::isTrusted () const

Test if the key is trusted.

Returns:

true if the key is trusted

10.51.3.12 QByteArray QCA::PGPKey::toArray () const

Export the key to an array.

This will export the key in a binary format (that is, not in an "ascii armoured" form).

See also:

[fromArray](#) for a static import method.

[toString](#) for an "ascii armoured" export method.

10.51.3.13 QString QCA::PGPKey::toString () const

Export the key to a string.

This will export the key in an "ascii armoured" form.

See also:

[fromString](#) for a static import method.

[toArray](#) for a binary format export method.

10.51.3.14 bool QCA::PGPKey::toFile (const QString & *fileName*) const

Export the key to a file.

Parameters:

fileName the name of the file to save the key to

10.51.3.15 static PGPKey QCA::PGPKey::fromArray (const QByteArray & *a*, ConvertResult * *result* = 0, const QString & *provider* = QString ()) [static]

Import the key from an array.

Parameters:

a the array to import from

result if not null, this will be set to the result of the import process

provider the provider to use, if a particular provider is required

10.51.3.16 static **PGPKey** QCA::PGPKey::fromString (const QString & *s*, **ConvertResult** * *result* = 0, const QString & *provider* = QString ()) [static]

Import the key from a string.

Parameters:

s the string to import from

result if not null, this will be set to the result of the import process

provider the provider to use, if a particular provider is required

10.51.3.17 static **PGPKey** QCA::PGPKey::fromFile (const QString & *fileName*, **ConvertResult** * *result* = 0, const QString & *provider* = QString ()) [static]

Import the key from a file.

Parameters:

fileName string containing the name of the file to import from

result if not null, this will be set to the result of the import process

provider the provider to use, if a particular provider is required

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

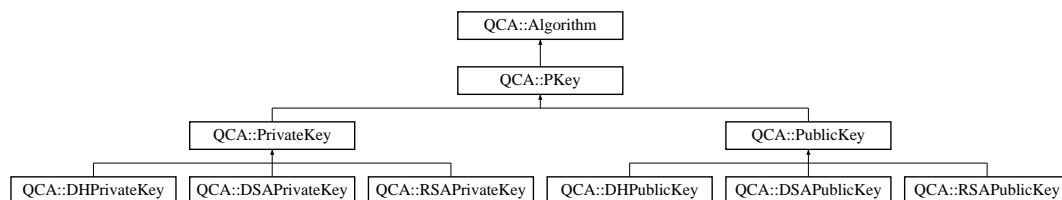
- [qca_cert.h](#)

10.52 QCA::PKey Class Reference

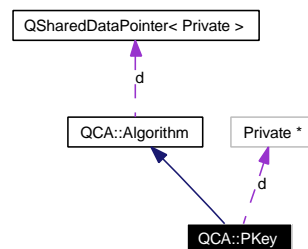
General superclass for public ([PublicKey](#)) and private ([PrivateKey](#)) keys used with asymmetric encryption techniques.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PKey::



Collaboration diagram for QCA::PKey:



Public Types

- enum [Type](#) { [RSA](#), [DSA](#), [DH](#) }

Public Member Functions

- [PKey](#) (const [PKey](#) &from)
- [PKey](#) & [operator=](#) (const [PKey](#) &from)
- bool [isNull](#) () const
- [Type](#) [type](#) () const
- int [bitSize](#) () const
- bool [isRSA](#) () const
- bool [isDSA](#) () const
- bool [isDH](#) () const
- bool [isPublic](#) () const
- bool [isPrivate](#) () const
- bool [canExport](#) () const
- bool [canKeyAgree](#) () const
- [PublicKey](#) [toPublicKey](#) () const
- [PrivateKey](#) [toPrivateKey](#) () const
- bool [operator==](#) (const [PKey](#) &a) const
- bool [operator!=](#) (const [PKey](#) &a) const

Static Public Member Functions

- static `QList< Type > supportedTypes` (const `QString` &provider=`QString()`)
- static `QList< Type > supportedIOTypes` (const `QString` &provider=`QString()`)

Protected Member Functions

- `PKey` (const `QString` &type, const `QString` &provider)
- void `set` (const `PKey` &k)
- `RSAPublicKey toRSAPublicKey` () const
- `RSAPrivateKey toRSAPrivateKey` () const
- `DSAPublicKey toDSAPublicKey` () const
- `DSAPrivateKey toDSAPrivateKey` () const
- `DHPublicKey toDHPublicKey` () const
- `DHPrivateKey toDHPrivateKey` () const

10.52.1 Detailed Description

General superclass for public (`PublicKey`) and private (`PrivateKey`) keys used with asymmetric encryption techniques.

10.52.2 Member Enumeration Documentation

10.52.2.1 enum `QCA::PKey::Type`

Types of public key cryptography keys supported by `QCA`.

Enumerator:

- RSA*** RSA key.
- DSA*** DSA key.
- DH*** Diffie Hellman key.

10.52.3 Constructor & Destructor Documentation

10.52.3.1 `QCA::PKey::PKey` (const `PKey` &*from*)

Standard copy constructor.

Parameters:

from the key to copy from

10.52.3.2 `QCA::PKey::PKey` (const `QString` &*type*, const `QString` &*provider*) [protected]

Create a key of the specified type.

10.52.4 Member Function Documentation

10.52.4.1 PKey & QCA::PKey::operator= (const PKey & *from*)

Standard assignment operator.

Parameters:

from the PKey to copy from

10.52.4.2 static QList<Type> QCA::PKey::supportedTypes (const QString & *provider* = QString()) [static]

Test what types of keys are supported.

Normally you would just test if the capability is present, however for PKey, you also need to test which types of keys are available. So if you want to figure out if RSA keys are supported, you need to do something like:

```
if(!QCA::isSupported("pkey") ||
    !QCA::PKey::supportedTypes().contains(QCA::PKey::RSA))
{
    // then there is no RSA key support
}
else
{
    // there is RSA key support
}
```

To make things a bit more complex, supportedTypes() only checks for basic functionality. If you want to check that you can do operations with PEM or DER (eg toPEM(), fromPEM(), and the equivalent DER and PEMfile operations, plus anything else that uses them, including the constructor form that takes a fileName), then you need to check for supportedIOTypes() instead.

See also:

supportedIOTypes()

10.52.4.3 static QList<Type> QCA::PKey::supportedIOTypes (const QString & *provider* = QString()) [static]

Test what types of keys are supported for IO operations.

If you are using PKey DER or PEM operations, then you need to check for appropriate support using this method. For example, if you want to check if you can export or import an RSA key, then you need to do something like:

```
if(!QCA::isSupported("pkey") ||
    !QCA::PKey::supportedIOTypes().contains(QCA::PKey::RSA))
{
    // then there is no RSA key IO support
}
else
{
    // there is RSA key IO support
}
```

Note that if you only want to check for basic functionality (ie not PEM or DER import/export), then you can use [supportedTypes\(\)](#). There is no need to use both - if the key type is supported for IO, then is also supported for basic operations.

See also:

[supportedTypes\(\)](#)

Examples:

[rsatest.cpp](#).

10.52.4.4 bool QCA::PKey::isNull () const

Test if the key is null (empty).

Returns:

true if the key is null

Examples:

[rsatest.cpp](#).

10.52.4.5 Type QCA::PKey::type () const

Report the Type of key (eg RSA, DSA or Diffie Hellman).

See also:

[isRSA](#), [isDSA](#) and [isDH](#) for boolean tests.

Reimplemented from [QCA::Algorithm](#).

10.52.4.6 int QCA::PKey::bitSize () const

Report the number of bits in the key.

10.52.4.7 bool QCA::PKey::isRSA () const

Test if the key is an RSA key.

10.52.4.8 bool QCA::PKey::isDSA () const

Test if the key is a DSA key.

10.52.4.9 bool QCA::PKey::isDH () const

Test if the key is a Diffie Hellman key.

10.52.4.10 bool QCA::PKey::isPublic () const

Test if the key is a public key.

10.52.4.11 bool QCA::PKey::isPrivate () const

Test if the key is a private key.

10.52.4.12 bool QCA::PKey::canExport () const

Test if the key data can be exported.

If the key resides on a smart card or other such device, this will likely return false.

10.52.4.13 bool QCA::PKey::canKeyAgree () const

Test if the key can be used for key agreement.

10.52.4.14 PublicKey QCA::PKey::toPublicKey () const

Interpret this key as a [PublicKey](#).

See also:

[toRSAPublicKey\(\)](#), [toDSAPublicKey\(\)](#) and [toDHPrivateKey\(\)](#) for protected forms of this call.

10.52.4.15 PrivateKey QCA::PKey::toPrivateKey () const

Interpret this key as a [PrivateKey](#).

10.52.4.16 bool QCA::PKey::operator== (const PKey & a) const

test if two keys are equal

10.52.4.17 bool QCA::PKey::operator!= (const PKey & a) const

test if two keys are not equal

10.52.4.18 void QCA::PKey::set (const PKey & k) [protected]

Set the key.

10.52.4.19 RSAPublicKey QCA::PKey::toRSAPublicKey () const [protected]

Interpret this key as an [RSAPublicKey](#).

Note:

This function is essentially a convenience cast - if the key was created as a DSA key, this function cannot turn it into an RSA key.

See also:

[toPublicKey\(\)](#) for the public version of this method

10.52.4.20 RSAPrivateKey QCA::PKey::toRSAPrivateKey () const [protected]

Interpret this key as an [RSAPrivateKey](#).

Note:

This function is essentially a convenience cast - if the key was created as a DSA key, this function cannot turn it into a RSA key.

See also:

[toPrivateKey\(\)](#) for the public version of this method

10.52.4.21 DSAPublicKey QCA::PKey::toDSAPublicKey () const [protected]

Interpret this key as an [DSAPublicKey](#).

Note:

This function is essentially a convenience cast - if the key was created as an RSA key, this function cannot turn it into a DSA key.

See also:

[toPublicKey\(\)](#) for the public version of this method

10.52.4.22 DSAPrivateKey QCA::PKey::toDSAPrivateKey () const [protected]

Interpret this key as a [DSAPrivateKey](#).

Note:

This function is essentially a convenience cast - if the key was created as an RSA key, this function cannot turn it into a DSA key.

See also:

[toPrivateKey\(\)](#) for the public version of this method

10.52.4.23 DHPublicKey QCA::PKey::toDHPublicKey () const [protected]

Interpret this key as an [DHPublicKey](#).

Note:

This function is essentially a convenience cast - if the key was created as a DSA key, this function cannot turn it into a DH key.

See also:

[toPublicKey\(\)](#) for the public version of this method

10.52.4.24 DHPrivateKey QCA::PKey::toDHPrivateKey () const [protected]

Interpret this key as a [DHPrivateKey](#).

Note:

This function is essentially a convenience cast - if the key was created as a DSA key, this function cannot turn it into a DH key.

See also:

[toPrivateKey\(\)](#) for the public version of this method

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

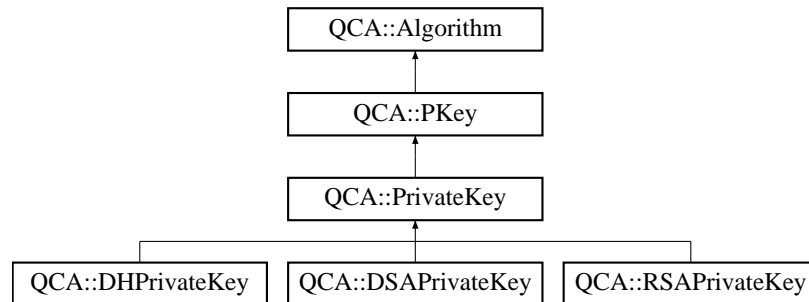
- [qca_publickey.h](#)

10.53 QCA::PrivateKey Class Reference

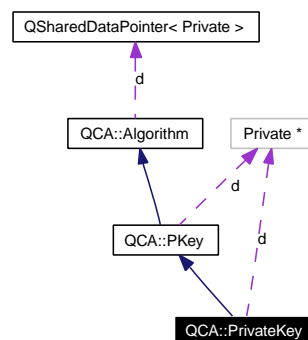
Generic private key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PrivateKey::



Collaboration diagram for QCA::PrivateKey:



Public Member Functions

- [PrivateKey](#) ()
- [PrivateKey](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase=[SecureArray](#)())
- [PrivateKey](#) (const [PrivateKey](#) &from)
- [PrivateKey](#) & operator= (const [PrivateKey](#) &from)
- [RSAPrivateKey toRSA](#) () const
- [DSAPrivateKey toDSA](#) () const
- [DHPrivateKey toDH](#) () const
- bool [canDecrypt](#) () const
- bool [canSign](#) () const
- bool [decrypt](#) (const [SecureArray](#) &in, [SecureArray](#) *out, [EncryptionAlgorithm](#) alg)
- void [startSign](#) ([SignatureAlgorithm](#) alg, [SignatureFormat](#) format=DefaultFormat)
- void [update](#) (const [MemoryRegion](#) &a)
- [QByteArray](#) [signature](#) ()
- [QByteArray](#) [signMessage](#) (const [MemoryRegion](#) &a, [SignatureAlgorithm](#) alg, [SignatureFormat](#) format=DefaultFormat)
- [SymmetricKey](#) [deriveKey](#) (const [PublicKey](#) &theirs)

- [SecureArray toDER](#) (const [SecureArray](#) &passphrase=[SecureArray](#)(), [PBEAlgorithm](#) pbe=[PBEDefault](#)) const
- [QString toPEM](#) (const [SecureArray](#) &passphrase=[SecureArray](#)(), [PBEAlgorithm](#) pbe=[PBEDefault](#)) const
- bool [toPEMFile](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase=[SecureArray](#)(), [PBEAlgorithm](#) pbe=[PBEDefault](#)) const

Static Public Member Functions

- static [QList< PBEAlgorithm > supportedPBEAlgorithms](#) (const [QString](#) &provider=[QString](#)())
- static [PrivateKey fromDER](#) (const [SecureArray](#) &a, const [SecureArray](#) &passphrase=[SecureArray](#)(), [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [PrivateKey fromPEM](#) (const [QString](#) &s, const [SecureArray](#) &passphrase=[SecureArray](#)(), [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())
- static [PrivateKey fromPEMFile](#) (const [QString](#) &fileName, const [SecureArray](#) &passphrase=[SecureArray](#)(), [ConvertResult](#) *result=0, const [QString](#) &provider=[QString](#)())

Protected Member Functions

- [PrivateKey](#) (const [QString](#) &type, const [QString](#) &provider)

10.53.1 Detailed Description

Generic private key.

Examples:

[cmsexample.cpp](#), [keyloader.cpp](#), [publickeyexample.cpp](#), [rsatest.cpp](#), and [sslservtest.cpp](#).

10.53.2 Constructor & Destructor Documentation

10.53.2.1 QCA::PrivateKey::PrivateKey ()

Create an empty private key.

10.53.2.2 QCA::PrivateKey::PrivateKey (const [QString](#) & *fileName*, const [SecureArray](#) & *passphrase* = [SecureArray](#) ()) [explicit]

Import a private key from a PEM representation in a file.

Parameters:

fileName the name of the file containing the private key

passphrase the pass phrase for the private key

See also:

[fromPEMFile](#) for an alternative method

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.53.2.3 QCA::PrivateKey::PrivateKey (const PrivateKey & *from*)

Copy constructor.

Parameters:

from the PrivateKey to copy from

10.53.2.4 QCA::PrivateKey::PrivateKey (const QString & *type*, const QString & *provider*) [protected]

Create a new private key.

Parameters:

type the type of key to create

provider the provider to use, if a specific provider is required.

10.53.3 Member Function Documentation

10.53.3.1 PrivateKey & QCA::PrivateKey::operator= (const PrivateKey & *from*)

Assignment operator.

Parameters:

from the PrivateKey to copy from

10.53.3.2 RSAPrivateKey QCA::PrivateKey::toRSA () const

Interpret / convert the key to an RSA key.

10.53.3.3 DSAPrivateKey QCA::PrivateKey::toDSA () const

Interpret / convert the key to a DSA key.

10.53.3.4 DHPrivateKey QCA::PrivateKey::toDH () const

Interpret / convert the key to a Diffie-Hellman key.

10.53.3.5 bool QCA::PrivateKey::canDecrypt () const

Test if this key can be used for decryption.

Returns:

true if the key can be used for decryption

Examples:

[publickeyexample.cpp](#).

10.53.3.6 bool QCA::PrivateKey::canSign () const

Test if this key can be used for signing.

Returns:

true if the key can be used to make a signature

Examples:

[rsatest.cpp](#).

10.53.3.7 bool QCA::PrivateKey::decrypt (const SecureArray & *in*, SecureArray * *out*, EncryptionAlgorithm *alg*)

Decrypt the message.

Parameters:

in the cipher (encrypted) data

out the plain text data

alg the algorithm to use

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[publickeyexample.cpp](#), and [rsatest.cpp](#).

10.53.3.8 void QCA::PrivateKey::startSign (SignatureAlgorithm *alg*, SignatureFormat *format* = DefaultFormat)

Initialise the message signature process.

Parameters:

alg the algorithm to use for the message signature process

format the signature format to use, for DSA

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[rsatest.cpp](#).

10.53.3.9 void QCA::PrivateKey::update (const MemoryRegion & *a*)

Update the signature process.

Parameters:

a the message to use to update the signature

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[rsatest.cpp](#).

10.53.3.10 QByteArray QCA::PrivateKey::signature ()

The resulting signature.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[rsatest.cpp](#).

10.53.3.11 QByteArray QCA::PrivateKey::signMessage (const [MemoryRegion](#) & *a*, [SignatureAlgorithm](#) *alg*, [SignatureFormat](#) *format* = DefaultFormat)

One step signature process.

Parameters:

a the message to sign
alg the algorithm to use for the signature
format the signature format to use, for DSA

Returns:

the signature

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.53.3.12 [SymmetricKey](#) QCA::PrivateKey::deriveKey (const [PublicKey](#) & *theirs*)

Derive a shared secret key from a public key.

Parameters:

theirs the public key to derive from

10.53.3.13 static QList<[PBEAlgorithm](#)> QCA::PrivateKey::supportedPBEAlgorithms (const QString & *provider* = QString()) [static]

List the supported Password Based Encryption Algorithms that can be used to protect the key.

Parameters:

provider the provider to use, if a particular provider is required

10.53.3.14 [SecureArray](#) QCA::PrivateKey::toDER (const [SecureArray](#) & *passphrase* = [SecureArray](#) (), [PBEAlgorithm](#) *pbe* = PBEDefault) const

Export the key in Distinguished Encoding Rules (DER) format.

Parameters:

passphrase the pass phrase to use to protect the key

pbe the symmetric encryption algorithm to use to protect the key

See also:

[fromDER](#) provides an inverse of [toDER](#), converting the DER encoded key back to a [PrivateKey](#)

10.53.3.15 [QString](#) QCA::PrivateKey::toPEM (const [SecureArray](#) & *passphrase* = [SecureArray](#) (), [PBEAlgorithm](#) *pbe* = PBEDefault) const

Export the key in Privacy Enhanced Mail (PEM) format.

Parameters:

passphrase the pass phrase to use to protect the key

pbe the symmetric encryption algorithm to use to protect the key

See also:

[toPEMFile](#) provides a convenient way to save the PEM encoded key to a file

[fromPEM](#) provides an inverse of [toPEM](#), converting the PEM encoded key back to a [PrivateKey](#)

10.53.3.16 [bool](#) QCA::PrivateKey::toPEMFile (const [QString](#) & *fileName*, const [SecureArray](#) & *passphrase* = [SecureArray](#) (), [PBEAlgorithm](#) *pbe* = PBEDefault) const

Export the key in Privacy Enhanced Mail (PEM) format to a file.

Parameters:

fileName the name (and path, if required) that the key should be exported to.

passphrase the pass phrase to use to protect the key

pbe the symmetric encryption algorithm to use to protect the key

Returns:

true if the export succeeds

See also:

[toPEM](#) provides a convenient way to save the PEM encoded key to a file

[fromPEM](#) provides an inverse of [toPEM](#), converting the PEM encoded key back to a [PrivateKey](#)

10.53.3.17 [static PrivateKey](#) QCA::PrivateKey::fromDER (const [SecureArray](#) & *a*, const [SecureArray](#) & *passphrase* = [SecureArray](#) (), [ConvertResult](#) * *result* = 0, const [QString](#) & *provider* = [QString](#) ()) [static]

Import the key from Distinguished Encoding Rules (DER) format.

Parameters:

- a* the array containing the DER representation of the key
- passphrase* the pass phrase that is used to protect the key
- result* a pointer to a `ConvertResult`, that if specified, will be set to reflect the result of the import
- provider* the provider to use, if a particular provider is required

See also:

[toDER](#) provides an inverse of [fromDER](#), exporting the key to an array
[QCA::KeyLoader](#) for an asynchronous loader approach.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.53.3.18 static [PrivateKey](#) `QCA::PrivateKey::fromPEM (const QString & s, const SecureArray & passphrase = SecureArray (), ConvertResult * result = 0, const QString & provider = QString ())` [static]

Import the key from Privacy Enhanced Mail (PEM) format.

Parameters:

- s* the string containing the PEM representation of the key
- passphrase* the pass phrase that is used to protect the key
- result* a pointer to a `ConvertResult`, that if specified, will be set to reflect the result of the import
- provider* the provider to use, if a particular provider is required

See also:

[toPEM](#) provides an inverse of [fromPEM](#), exporting the key to a string in PEM encoding.
[QCA::KeyLoader](#) for an asynchronous loader approach.

Note:

This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[sslservtest.cpp](#).

10.53.3.19 static [PrivateKey](#) `QCA::PrivateKey::fromPEMFile (const QString & fileName, const SecureArray & passphrase = SecureArray (), ConvertResult * result = 0, const QString & provider = QString ())` [static]

Import the key in Privacy Enhanced Mail (PEM) format from a file.

Parameters:

- fileName* the name (and path, if required) of the file containing the PEM representation of the key
- passphrase* the pass phrase that is used to protect the key
- result* a pointer to a `ConvertResult`, that if specified, will be set to reflect the result of the import
- provider* the provider to use, if a particular provider is required

See also:

[toPEMFile](#) provides an inverse of [fromPEMFile](#)
[fromPEM](#) which allows import from a string
[QCA::KeyLoader](#) for an asynchronous loader approach.

Note:

there is also a constructor form, that allows you to create the key directly
This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

Examples:

[cmsexample.cpp](#), [publickeyexample.cpp](#), and [rsatest.cpp](#).

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

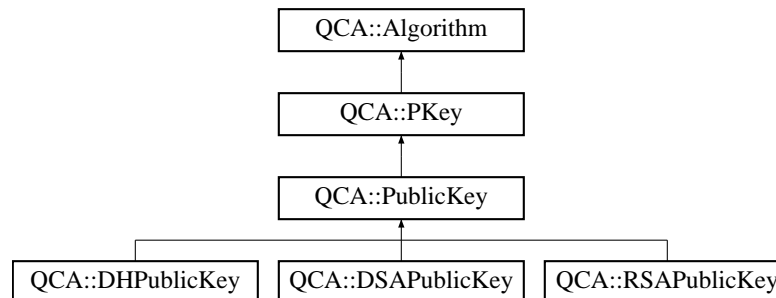
- [qca_publickey.h](#)

10.54 QCA::PublicKey Class Reference

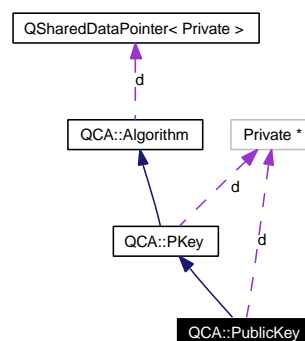
Generic public key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::PublicKey::



Collaboration diagram for QCA::PublicKey:



Public Member Functions

- [PublicKey](#) ()
- [PublicKey](#) (const [PrivateKey](#) &k)
- [PublicKey](#) (const **QString** &fileName)
- [PublicKey](#) (const [PublicKey](#) &from)
- [PublicKey](#) & [operator=](#) (const [PublicKey](#) &from)
- [RSAPublicKey toRSA](#) () const
- [DSAPublicKey toDSA](#) () const
- [DHPublicKey toDH](#) () const
- bool [canEncrypt](#) () const
- bool [canVerify](#) () const
- int [maximumEncryptSize](#) ([EncryptionAlgorithm](#) alg) const
- [SecureArray encrypt](#) (const [SecureArray](#) &a, [EncryptionAlgorithm](#) alg)
- void [startVerify](#) ([SignatureAlgorithm](#) alg, [SignatureFormat](#) format=DefaultFormat)
- void [update](#) (const [MemoryRegion](#) &a)
- bool [validSignature](#) (const **QByteArray** &sig)

- bool [verifyMessage](#) (const [MemoryRegion](#) &a, const [QByteArray](#) &sig, [SignatureAlgorithm](#) alg, [SignatureFormat](#) format=DefaultFormat)
- [QByteArray toDER](#) () const
- [QString toPEM](#) () const
- bool [toPEMFile](#) (const [QString](#) &fileName) const

Static Public Member Functions

- static [PublicKey fromDER](#) (const [QByteArray](#) &a, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString\(\)](#))
- static [PublicKey fromPEM](#) (const [QString](#) &s, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString\(\)](#))
- static [PublicKey fromPEMFile](#) (const [QString](#) &fileName, [ConvertResult](#) *result=0, const [QString](#) &provider=[QString\(\)](#))

Protected Member Functions

- [PublicKey](#) (const [QString](#) &type, const [QString](#) &provider)

10.54.1 Detailed Description

Generic public key.

Examples:

[rsatest.cpp](#).

10.54.2 Constructor & Destructor Documentation

10.54.2.1 QCA::PublicKey::PublicKey ()

Create an empty (null) public key.

10.54.2.2 QCA::PublicKey::PublicKey (const [PrivateKey](#) &*k*)

Create a public key based on a specified private key.

Parameters:

k the private key to extract the public key parts from

10.54.2.3 QCA::PublicKey::PublicKey (const [QString](#) &*fileName*)

Import a public key from a PEM representation in a file.

Parameters:

fileName the name of the file containing the public key

See also:

[fromPEMFile](#) for an alternative method

10.54.2.4 QCA::PublicKey::PublicKey (const [PublicKey](#) & *from*)

Copy constructor.

Parameters:

from the [PublicKey](#) to copy from

10.54.2.5 QCA::PublicKey::PublicKey (const QString & *type*, const QString & *provider*) [protected]

Create a new key of a specified type.

Parameters:

type the type of key to create

provider the provider to use, if required

10.54.3 Member Function Documentation

10.54.3.1 [PublicKey](#)& QCA::PublicKey::operator= (const [PublicKey](#) & *from*)

Assignment operator.

Parameters:

from the [PublicKey](#) to copy from

10.54.3.2 [RSAPublicKey](#) QCA::PublicKey::toRSA () const

Convenience method to convert this key to an [RSAPublicKey](#).

Note that if the key is not an RSA key (eg it is DSA or DH), then this will produce a null key.

10.54.3.3 [DSAPublicKey](#) QCA::PublicKey::toDSA () const

Convenience method to convert this key to a [DSAPublicKey](#).

Note that if the key is not an DSA key (eg it is RSA or DH), then this will produce a null key.

10.54.3.4 [DHPublicKey](#) QCA::PublicKey::toDH () const

Convenience method to convert this key to a [DHPublicKey](#).

Note that if the key is not an DH key (eg it is DSA or RSA), then this will produce a null key.

10.54.3.5 bool QCA::PublicKey::canEncrypt () const

Test if this key can be used for encryption.

Returns:

true if the key can be used for encryption

Examples:

[rsatest.cpp](#).

10.54.3.6 bool QCA::PublicKey::canVerify () const

Test if the key can be used for verifying signatures.

Returns:

true of the key can be used for verification

10.54.3.7 int QCA::PublicKey::maximumEncryptSize (EncryptionAlgorithm *alg*) const

The maximum message size that can be encrypted with a specified algorithm.

Parameters:

alg the algorithm to check

10.54.3.8 SecureArray QCA::PublicKey::encrypt (const SecureArray & *a*, EncryptionAlgorithm *alg*)

Encrypt a message using a specified algorithm.

Parameters:

a the message to encrypt

alg the algorithm to use

10.54.3.9 void QCA::PublicKey::startVerify (SignatureAlgorithm *alg*, SignatureFormat *format* = DefaultFormat)

Initialise the signature verification process.

Parameters:

alg the algorithm to use for signing

format the specific format to use, for DSA

10.54.3.10 void QCA::PublicKey::update (const MemoryRegion & *a*)

Update the signature verification process with more data.

Parameters:

a the array containing the data that should be added to the signature

10.54.3.11 bool QCA::PublicKey::validSignature (const QByteArray & sig)

Check the signature is valid for the message.

The process to check that a signature is correct is shown below:

```
// note that pubkey is a PublicKey
if( pubkey.canVerify() )
{
    pubkey.startVerify( QCA::EMSA3_MD5 );
    pubkey.update( theMessage ); // might be called multiple times
    if ( pubkey.validSignature( theSignature ) )
    {
        // then signature is valid
    }
    else
    {
        // then signature is invalid
    }
}
```

Parameters:

sig the signature to check

Returns:

true if the signature is correct

10.54.3.12 bool QCA::PublicKey::verifyMessage (const [MemoryRegion](#) & a, const QByteArray & sig, [SignatureAlgorithm](#) alg, [SignatureFormat](#) format = DefaultFormat)

Single step message verification.

If you have the whole message to be verified, then this offers a more convenient approach to verification.

Parameters:

a the message to check the signature on

sig the signature to be checked

alg the algorithm to use

format the signature format to use, for DSA

Returns:

true if the signature is valid for the message

10.54.3.13 QByteArray QCA::PublicKey::toDER () const

Export the key in Distinguished Encoding Rules (DER) format.

10.54.3.14 QString QCA::PublicKey::toPEM () const

Export the key in Privacy Enhanced Mail (PEM) format.

See also:

[toPEMFile](#) provides a convenient way to save the PEM encoded key to a file

[fromPEM](#) provides an inverse of [toPEM](#), converting the PEM encoded key back to a [PublicKey](#)

10.54.3.15 `bool QCA::PublicKey::toPEMFile (const QString & fileName) const`

Export the key in Privacy Enhanced Mail (PEM) to a file.

Parameters:

fileName the name (and path, if necessary) of the file to save the PEM encoded key to.

See also:

[toPEM](#) for a version that exports to a **QString**, which may be useful if you need to do more sophisticated handling

[fromPEMFile](#) provides an inverse of [toPEMFile](#), reading a PEM encoded key from a file

10.54.3.16 `static PublicKey QCA::PublicKey::fromDER (const QByteArray & a, ConvertResult * result = 0, const QString & provider = QString()) [static]`

Import a key in Distinguished Encoding Rules (DER) format.

This function takes a binary array, which is assumed to contain a public key in DER encoding, and returns the key. Unless you don't care whether the import succeeded, you should test the result, as shown below.

```
QCA::ConvertResult conversionResult;
QCA::PublicKey publicKey = QCA::PublicKey::fromDER(keyArray, &conversionResult);
if (! QCA::ConvertGood == conversionResult)
{
    std::cout << "Public key read failed" << std::endl;
}
```

Parameters:

a the array containing a DER encoded key

result pointer to a variable, which returns whether the conversion succeeded (ConvertGood) or not

provider the name of the provider to use for the import.

10.54.3.17 `static PublicKey QCA::PublicKey::fromPEM (const QString & s, ConvertResult * result = 0, const QString & provider = QString()) [static]`

Import a key in Privacy Enhanced Mail (PEM) format.

This function takes a string, which is assumed to contain a public key in PEM encoding, and returns that key. Unless you don't care whether the import succeeded, you should test the result, as shown below.

```
QCA::ConvertResult conversionResult;
QCA::PublicKey publicKey = QCA::PublicKey::fromPEM(keyAsString, &conversionResult);
if (! QCA::ConvertGood == conversionResult)
{
    std::cout << "Public key read failed" << std::endl;
}
```

Parameters:

s the string containing a PEM encoded key

result pointer to a variable, which returns whether the conversion succeeded (ConvertGood) or not

provider the name of the provider to use for the import.

See also:

[toPEM](#), which provides an inverse of [fromPEM\(\)](#)

[fromPEMFile](#), which provides an import direct from a file.

10.54.3.18 static **PublicKey** QCA::PublicKey::fromPEMFile (const QString & *fileName*, **ConvertResult** * *result* = 0, const QString & *provider* = QString()) [static]

Import a key in Privacy Enhanced Mail (PEM) format from a file.

This function takes the name of a file, which is assumed to contain a public key in PEM encoding, and returns that key. Unless you don't care whether the import succeeded, you should test the result, as shown below.

```
QCA::ConvertResult conversionResult;  
QCA::PublicKey publicKey = QCA::PublicKey::fromPEMFile(fileName, &conversionResult);  
if (! QCA::ConvertGood == conversionResult)  
{  
    std::cout << "Public key read failed" << std::endl;  
}
```

Parameters:

fileName a string containing the name of the file

result pointer to a variable, which returns whether the conversion succeeded (ConvertGood) or not

provider the name of the provider to use for the import.

See also:

[toPEMFile](#), which provides an inverse of [fromPEMFile\(\)](#)
[fromPEM](#), which provides an import from a string

Note:

there is also a constructor form that can import from a file

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

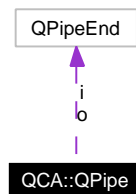
- [qca_publickey.h](#)

10.55 QCA::QPipe Class Reference

A FIFO buffer (named pipe) abstraction.

```
#include <qpipe.h>
```

Collaboration diagram for QCA::QPipe:



Public Member Functions

- [QPipe](#) ([QObject](#) *parent=0)
- void [reset](#) ()
- bool [create](#) ()
- [QPipeEnd](#) & [readEnd](#) ()
- [QPipeEnd](#) & [writeEnd](#) ()

10.55.1 Detailed Description

A FIFO buffer (named pipe) abstraction.

This class creates a full buffer, consisting of two ends ([QPipeEnd](#)).

By default, the pipe ends are not inheritable by child processes. On Windows, the pipe is created with inheritability disabled. On Unix, the `FD_CLOEXEC` flag is set on each end's file descriptor.

10.55.2 Constructor & Destructor Documentation

10.55.2.1 QCA::QPipe::QPipe (QObject *parent = 0)

Standard constructor.

Parameters:

parent the parent object for this object

10.55.3 Member Function Documentation

10.55.3.1 void QCA::QPipe::reset ()

reset the pipe

10.55.3.2 bool QCA::QPipe::create ()

create the pipe

10.55.3.3 QPipeEnd& QCA::QPipe::readEnd () [inline]

The read end of the pipe.

10.55.3.4 QPipeEnd& QCA::QPipe::writeEnd () [inline]

The write end of the pipe.

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

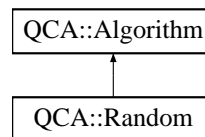
- [qpipe.h](#)

10.56 QCA::Random Class Reference

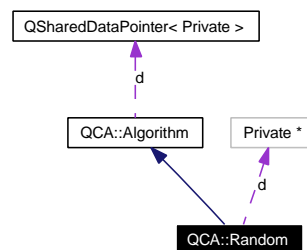
Source of random numbers.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::Random::



Collaboration diagram for QCA::Random:



Public Member Functions

- [Random](#) (const [QString](#) &provider=[QString](#)())
- [Random](#) (const [Random](#) &from)
- [Random](#) & [operator=](#) (const [Random](#) &from)
- uchar [nextByte](#) ()
- [SecureArray](#) [nextBytes](#) (int size)

Static Public Member Functions

- static uchar [randomChar](#) ()
- static int [randomInt](#) ()
- static [SecureArray](#) [randomArray](#) (int size)

10.56.1 Detailed Description

Source of random numbers.

[QCA](#) provides a built in source of random numbers, which can be accessed through this class. You can also use an alternative random number source, by implementing another provider.

The normal use of this class is expected to be through the static members - [randomChar\(\)](#), [randomInt\(\)](#) and [randomArray\(\)](#).

Examples:

[randomtest.cpp](#).

10.56.2 Constructor & Destructor Documentation

10.56.2.1 QCA::Random::Random (const QString & *provider* = QString ())

Standard Constructor.

Parameters:

provider the name of the provider library for the random number generation

10.56.2.2 QCA::Random::Random (const Random & *from*)

Copy constructor.

Parameters:

from the [Random](#) object to copy from

10.56.3 Member Function Documentation

10.56.3.1 Random& QCA::Random::operator= (const Random & *from*)

Assignment operator.

Parameters:

from the [Random](#) object to copy state from

10.56.3.2 uchar QCA::Random::nextByte ()

Provide a random byte.

This method isn't normally required - you should use the static [randomChar\(\)](#) method instead.

See also:

[randomChar](#)

Examples:

[randomtest.cpp](#).

10.56.3.3 SecureArray QCA::Random::nextBytes (int *size*)

Provide a specified number of random bytes.

This method isn't normally required - you should use the static [randomArray\(\)](#) method instead.

Parameters:

size the number of bytes to provide

See also:

[randomArray](#)

Examples:

[randomtest.cpp](#).

10.56.3.4 static uchar QCA::Random::randomChar () [static]

Provide a random character (byte).

This is the normal way of obtaining a single random char (ie. 8 bit byte), as shown below:

```
myRandomChar = QCA::Random::randomChar();
```

If you need a number of bytes, perhaps [randomArray\(\)](#) may be of use

Examples:

[randomtest.cpp](#).

10.56.3.5 static int QCA::Random::randomInt () [static]

Provide a random integer.

This is the normal way of obtaining a single random integer, as shown below:

```
myRandomInt = QCA::Random::randomInt();
```

Examples:

[randomtest.cpp](#).

10.56.3.6 static SecureArray QCA::Random::randomArray (int *size*) [static]

Provide a specified number of random bytes.

```
// build a 30 byte secure array.  
SecureArray array = QCA::Random::randomArray(30);
```

Parameters:

size the number of bytes to provide

Examples:

[randomtest.cpp](#).

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

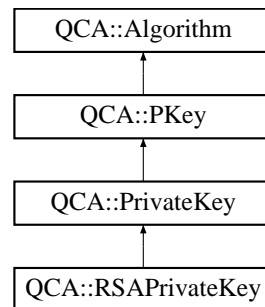
- [qca_basic.h](#)

10.57 QCA::RSAPrivateKey Class Reference

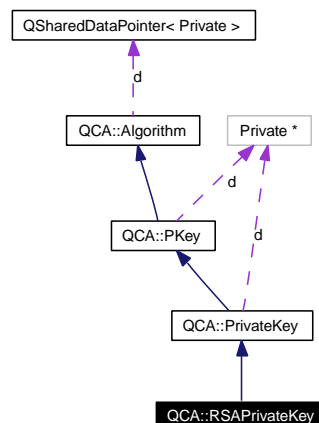
RSA Private Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::RSAPrivateKey::



Collaboration diagram for QCA::RSAPrivateKey:



Public Member Functions

- `RSAPrivateKey ()`
- `RSAPrivateKey (const BigInteger &n, const BigInteger &e, const BigInteger &p, const BigInteger &q, const BigInteger &d, const QString &provider=QString())`
- `BigInteger n () const`
- `BigInteger e () const`
- `BigInteger p () const`
- `BigInteger q () const`
- `BigInteger d () const`

10.57.1 Detailed Description

RSA Private Key.

10.57.2 Constructor & Destructor Documentation

10.57.2.1 QCA::RSAPrivateKey::RSAPrivateKey ()

Generate an empty RSA private key.

10.57.2.2 QCA::RSAPrivateKey::RSAPrivateKey (const [BigInteger](#) & *n*, const [BigInteger](#) & *e*, const [BigInteger](#) & *p*, const [BigInteger](#) & *q*, const [BigInteger](#) & *d*, const QString & *provider* = QString ())

Generate an RSA private key from specified parameters.

Parameters:

- n* the public key value
- e* the public key exponent
- p* one of the two chosen primes
- q* the other of the two chosen primes
- d* inverse of the exponent, modulo (p-1)(q-1)
- provider* the provider to use, if a particular provider is required

10.57.3 Member Function Documentation

10.57.3.1 [BigInteger](#) QCA::RSAPrivateKey::n () const

The public key value.

This value is the actual public key value (the product of p and q, the random prime numbers used to generate the RSA key), also known as the public modulus.

10.57.3.2 [BigInteger](#) QCA::RSAPrivateKey::e () const

The public key exponent.

This value is the exponent chosen in the original key generator step

10.57.3.3 [BigInteger](#) QCA::RSAPrivateKey::p () const

One of the two random primes used to generate the private key.

10.57.3.4 [BigInteger](#) QCA::RSAPrivateKey::q () const

The second of the two random primes used to generate the private key.

10.57.3.5 [BigInteger](#) QCA::RSAPrivateKey::d () const

The inverse of the exponent, module (p-1)(q-1).

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

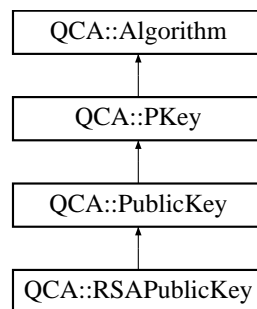
- [qca_publickey.h](#)

10.58 QCA::RSAPublicKey Class Reference

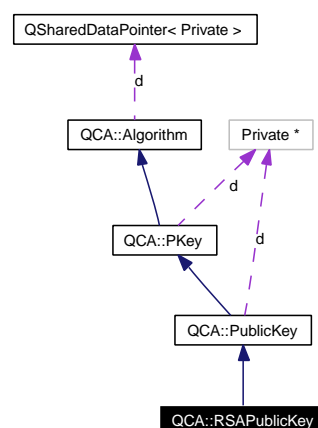
RSA Public Key.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::RSAPublicKey::



Collaboration diagram for QCA::RSAPublicKey:



Public Member Functions

- [RSAPublicKey \(\)](#)
- [RSAPublicKey \(const \[BigInteger\]\(#\) &n, const \[BigInteger\]\(#\) &e, const \[QString\]\(#\) &provider=\[QString\\(\\)\]\(#\)\)](#)
- [RSAPublicKey \(const \[RSAPrivateKey\]\(#\) &k\)](#)
- [BigInteger n \(\) const](#)
- [BigInteger e \(\) const](#)

10.58.1 Detailed Description

RSA Public Key.

10.58.2 Constructor & Destructor Documentation

10.58.2.1 QCA::RSAPublicKey::RSAPublicKey ()

Generate an empty RSA public key.

10.58.2.2 QCA::RSAPublicKey::RSAPublicKey (const [BigInteger](#) & *n*, const [BigInteger](#) & *e*, const [QString](#) & *provider* = [QString](#) ())

Generate an RSA public key from specified parameters.

Parameters:

n the public key value

e the public key exponent

provider the provider to use, if a particular provider is required

10.58.2.3 QCA::RSAPublicKey::RSAPublicKey (const [RSAPrivateKey](#) & *k*)

Extract the public key components from an RSA private key.

Parameters:

k the private key to use as the basis for the public key

10.58.3 Member Function Documentation

10.58.3.1 [BigInteger](#) QCA::RSAPublicKey::n () const

The public key value.

This value is the actual public key value (the product of p and q, the random prime numbers used to generate the RSA key), also known as the public modulus.

10.58.3.2 [BigInteger](#) QCA::RSAPublicKey::e () const

The public key exponent.

This value is the exponent chosen in the original key generator step

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

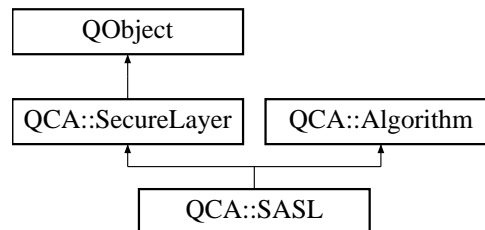
- [qca_publickey.h](#)

10.59 QCA::SASL Class Reference

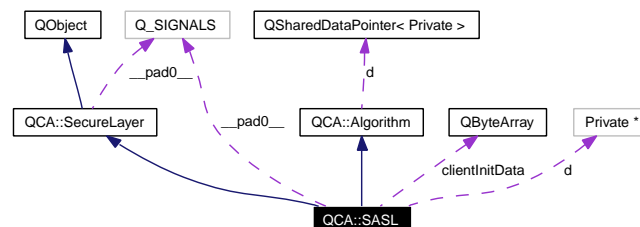
Simple Authentication and Security Layer protocol implementation.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SASL:



Collaboration diagram for QCA::SASL:



Public Types

- enum [Error](#) { [ErrorInit](#), [ErrorHandshake](#), [ErrorCrypt](#) }
- enum [AuthCondition](#) { [AuthFail](#), [NoMechanism](#), [BadProtocol](#), [BadServer](#), [BadAuth](#), [NoAuthzid](#), [TooWeak](#), [NeedEncrypt](#), [Expired](#), [Disabled](#), [NoUser](#), [RemoteUnavailable](#) }
- enum [AuthFlags](#) { [AuthFlagsNone](#) = 0x00, [AllowPlain](#) = 0x01, [AllowAnonymous](#) = 0x02, [RequireForwardSecrecy](#) = 0x04, [RequirePassCredentials](#) = 0x08, [RequireMutualAuth](#) = 0x10, [RequireAuthzidSupport](#) = 0x20 }
- enum [ClientSendMode](#) { [AllowClientSendFirst](#), [DisableClientSendFirst](#) }
- enum [ServerSendMode](#) { [AllowServerSendLast](#), [DisableServerSendLast](#) }

Public Member Functions

- [SASL](#) ([QObject](#) *parent=0, const [QString](#) &provider=[QString](#)())
- void [reset](#) ()
- void [setConstraints](#) ([AuthFlags](#) f, [SecurityLevel](#) s=[SL_None](#))
- void [setConstraints](#) ([AuthFlags](#) f, int minSSF, int maxSSF)
- void [setLocalAddress](#) (const [QString](#) &addr, quint16 port)
- void [setRemoteAddress](#) (const [QString](#) &addr, quint16 port)

- void [setExternalAuthId](#) (const **QString** &authid)
- void [setExternalSSF](#) (int strength)
- void [startClient](#) (const **QString** &service, const **QString** &host, const **QStringList** &mechlist, [ClientSendMode](#) mode=AllowClientSendFirst)
- void [startServer](#) (const **QString** &service, const **QString** &host, const **QString** &realm, [ServerSendMode](#) mode=DisableServerSendLast)
- void [putServerFirstStep](#) (const **QString** &mech)
- void [putServerFirstStep](#) (const **QString** &mech, const **QByteArray** &clientInit)
- void [putStep](#) (const **QByteArray** &stepData)
- **QString** [mechanism](#) () const
- **QStringList** [mechanismList](#) () const
- **QStringList** [realmList](#) () const
- int [ssf](#) () const
- **Error** [errorCode](#) () const
- **AuthCondition** [authCondition](#) () const
- void [setUsername](#) (const **QString** &user)
- void [setAuthzid](#) (const **QString** &auth)
- void [setPassword](#) (const [SecureArray](#) &pass)
- void [setRealm](#) (const **QString** &realm)
- void [continueAfterParams](#) ()
- void [continueAfterAuthCheck](#) ()
- virtual int [bytesAvailable](#) () const
- virtual int [bytesOutgoingAvailable](#) () const
- virtual void [write](#) (const **QByteArray** &a)
- virtual **QByteArray** [read](#) ()
- virtual void [writeIncoming](#) (const **QByteArray** &a)
- virtual **QByteArray** [readOutgoing](#) (int *plainBytes=0)
- virtual int [convertBytesWritten](#) (qint64 encryptedBytes)
- void [serverStarted](#) ()
- void [nextStep](#) (const **QByteArray** &stepData)
- void [needParams](#) (const [QCA::SASL::Params](#) ¶ms)
- void [authCheck](#) (const **QString** &user, const **QString** &authzid)
- void [authenticated](#) ()

Public Attributes

- Q_SIGNALS [__pad0__](#): void clientStarted(bool clientInit
- Q_SIGNALS const **QByteArray** & [clientInitData](#)

Friends

- class **Private**

Classes

- class [Params](#)

Parameter flags for the [SASL](#) authentication.

10.59.1 Detailed Description

Simple Authentication and Security Layer protocol implementation.

This class implements the Simple Authentication and Security Layer protocol, which is described in RFC2222 - see <http://www.ietf.org/rfc/rfc2222.txt>.

As the name suggests, SASL provides authentication (eg, a "login" of some form), for a connection oriented protocol, and can also provide protection for the subsequent connection.

The SASL protocol is designed to be extensible, through a range of "mechanisms", where a mechanism is the actual authentication method. Example mechanisms include Anonymous, LOGIN, Kerberos V4, and GSSAPI. Mechanisms can be added (potentially without restarting the server application) by the system administrator.

It is important to understand that SASL is neither "network aware" nor "protocol aware". That means that SASL does not understand how the client connects to the server, and SASL does not understand the actual application protocol.

Examples:

[saslservtest.cpp](#), and [sasctest.cpp](#).

10.59.2 Member Enumeration Documentation

10.59.2.1 enum [QCA::SASL::Error](#)

Possible errors that may occur when using [SASL](#).

Enumerator:

- ErrorInit* problem starting up [SASL](#)
- ErrorHandshake* problem during the authentication process
- ErrorCrypt* problem at anytime after

10.59.2.2 enum [QCA::SASL::AuthCondition](#)

Possible authentication error states.

Enumerator:

- AuthFail* Generic authentication failure.
- NoMechanism* No compatible/appropriate authentication mechanism.
- BadProtocol* Bad protocol or cancelled.
- BadServer* Server failed mutual authentication (client side only).
- BadAuth* Authentication failure (server side only).
- NoAuthzid* Authorization failure (server side only).
- TooWeak* Mechanism too weak for this user (server side only).
- NeedEncrypt* Encryption is needed in order to use mechanism (server side only).
- Expired* Passphrase expired, has to be reset (server side only).
- Disabled* Account is disabled (server side only).
- NoUser* User not found (server side only).
- RemoteUnavailable* Remote service needed for auth is gone (server side only).

10.59.2.3 enum [QCA::SASL::AuthFlags](#)

Authentication requirement flag values.

10.59.2.4 enum [QCA::SASL::ClientSendMode](#)

Mode options for client side sending.

10.59.2.5 enum [QCA::SASL::ServerSendMode](#)

Mode options for server side sending.

10.59.3 Constructor & Destructor Documentation

10.59.3.1 [QCA::SASL::SASL](#) ([QObject](#) * *parent* = 0, const [QString](#) & *provider* = [QString](#) ())

Standard constructor.

Parameters:

parent the parent object for this [SASL](#) connection

provider if specified, the provider to use. If not specified, or specified as empty, then any provider is acceptable.

10.59.4 Member Function Documentation

10.59.4.1 void [QCA::SASL::reset](#) ()

Reset the [SASL](#) mechanism.

10.59.4.2 void [QCA::SASL::setConstraints](#) ([AuthFlags](#) *f*, [SecurityLevel](#) *s* = [SL_None](#))

Specify connection constraints.

[SASL](#) supports a range of authentication requirements, and a range of security levels. This method allows you to specify the requirements for your connection.

Parameters:

f the authentication requirements, which you typically build using a binary OR function (eg `AllowPlain | AllowAnonymous`)

s the security level of the encryption, if used. See [SecurityLevel](#) for details of what each level provides.

Examples:

[saslservtest.cpp](#), and [sasctest.cpp](#).

10.59.4.3 void [QCA::SASL::setConstraints](#) ([AuthFlags](#) *f*, int *minSSF*, int *maxSSF*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Unless you have a specific reason for directly specifying a strength factor, you probably should use the method above.

Parameters:

f the authentication requirements, which you typically build using a binary OR function (eg Allow-Plain | AllowAnonymous)

minSSF the minimum security strength factor that is required

maxSSF the maximum security strength factor that is required

Note:

Security strength factors are a rough approximation to key length in the encryption function (eg if you are securing with plain DES, the security strength factor would be 56).

10.59.4.4 void QCA::SASL::setLocalAddress (const QString & *addr*, quint16 *port*)

Specify the local address.

Parameters:

addr the address of the local part of the connection

port the port number of the local part of the connection

10.59.4.5 void QCA::SASL::setRemoteAddress (const QString & *addr*, quint16 *port*)

Specify the peer address.

Parameters:

addr the address of the peer side of the connection

port the port number of the peer side of the connection

10.59.4.6 void QCA::SASL::setExternalAuthId (const QString & *authid*)

Specify the id of the externally secured connection.

Parameters:

authid the id of the connection

10.59.4.7 void QCA::SASL::setExternalSSF (int *strength*)

Specify a security strength factor for an externally secured connection.

Parameters:

strength the security strength factor of the connection

10.59.4.8 void QCA::SASL::startClient (const QString & *service*, const QString & *host*, const QStringList & *mechlist*, [ClientSendMode](#) *mode* = AllowClientSendFirst)

Initialise the client side of the connection.

startClient must be called on the client side of the connection. clientStarted will be emitted when the operation is completed.

Parameters:

- service* the name of the service
- host* the client side host name
- mechlist* the list of mechanisms which can be used
- mode* the mode to use on the client side

Examples:

[saslttest.cpp](#).

10.59.4.9 void QCA::SASL::startServer (const QString & *service*, const QString & *host*, const QString & *realm*, [ServerSendMode](#) *mode* = DisableServerSendLast)

Initialise the server side of the connection.

startServer must be called on the server side of the connection. serverStarted will be emitted when the operation is completed.

Parameters:

- service* the name of the service
- host* the server side host name
- realm* the realm to use
- mode* which mode to use on the server side

Examples:

[saslservtest.cpp](#).

10.59.4.10 void QCA::SASL::putServerFirstStep (const QString & *mech*)

Process the first step in server mode (server).

Call this with the mechanism selected by the client. If there is initial client data, call the other version of this function instead.

Examples:

[saslservtest.cpp](#).

10.59.4.11 void QCA::SASL::putServerFirstStep (const QString & *mech*, const QByteArray & *clientInit*)

Process the first step in server mode (server).

Call this with the mechanism selected by the client, and initial client data. If there is no initial client data, call the other version of this function instead.

10.59.4.12 void QCA::SASL::putStep (const QByteArray & *stepData*)

Process an authentication step.

Call this with authentication data received from the network. The only exception is the first step in server mode, in which case putServerFirstStep must be called.

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.13 QString QCA::SASL::mechanism () const

Return the mechanism selected (client).

Examples:

[saslttest.cpp](#).

10.59.4.14 QStringList QCA::SASL::mechanismList () const

Return the mechanism list (server).

Examples:

[saslservtest.cpp](#).

10.59.4.15 QStringList QCA::SASL::realmList () const

Return the realm list, if available (client).

Examples:

[saslttest.cpp](#).

10.59.4.16 int QCA::SASL::ssf () const

Return the security strength factor of the connection.

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.17 Error QCA::SASL::errorCode () const

Return the error code.

Examples:

[saslservtest.cpp](#).

10.59.4.18 AuthCondition QCA::SASL::authCondition () const

Return the reason for authentication failure.

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.19 void QCA::SASL::setUsername (const QString & user)

Specify the username to use in authentication.

Parameters:

user the username to use

Examples:

[saslttest.cpp](#).

10.59.4.20 void QCA::SASL::setAuthzid (const QString & auth)

Specify the authorization identity to use in authentication.

Parameters:

auth the authorization identity to use

Examples:

[saslttest.cpp](#).

10.59.4.21 void QCA::SASL::setPassword (const SecureArray & pass)

Specify the password to use in authentication.

Parameters:

pass the password to use

Examples:

[saslttest.cpp](#).

10.59.4.22 void QCA::SASL::setRealm (const QString & realm)

Specify the realm to use in authentication.

Parameters:

realm the realm to use

10.59.4.23 void QCA::SASL::continueAfterParams ()

Continue negotiation after parameters have been set (client).

10.59.4.24 void QCA::SASL::continueAfterAuthCheck ()

Continue negotiation after auth ids have been checked (server).

Examples:

[saslservtest.cpp](#).

10.59.4.25 virtual int QCA::SASL::bytesAvailable () const [virtual]

Returns the number of bytes available to be [read\(\)](#) on the application side.

Implements [QCA::SecureLayer](#).

10.59.4.26 virtual int QCA::SASL::bytesOutgoingAvailable () const [virtual]

Returns the number of bytes available to be [readOutgoing\(\)](#) on the network side.

Implements [QCA::SecureLayer](#).

10.59.4.27 virtual void QCA::SASL::write (const QByteArray & a) [virtual]

This method writes unencrypted (plain) data to the [SecureLayer](#) implementation.

You normally call this function on the application side.

Implements [QCA::SecureLayer](#).

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.28 virtual QByteArray QCA::SASL::read () [virtual]

This method reads decrypted (plain) data from the [SecureLayer](#) implementation.

You normally call this function on the application side after receiving the [readyRead\(\)](#) signal.

Implements [QCA::SecureLayer](#).

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.29 virtual void QCA::SASL::writeIncoming (const QByteArray & a) [virtual]

This method accepts encoded (typically encrypted) data for processing.

You normally call this function using data read from the network socket (e.g. using [QTcpSocket::readAll\(\)](#)) after receiving a signal that indicates that the socket has data to read.

Implements [QCA::SecureLayer](#).

Examples:

[saslttest.cpp](#).

10.59.4.30 virtual QByteArray QCA::SASL::readOutgoing (int * *plainBytes* = 0) [virtual]

This method provides encoded (typically encrypted) data.

You normally call this function to get data to write out to the network socket (e.g. using `QTcpSocket::write()`) after receiving the `readyReadOutgoing()` signal.

Implements [QCA::SecureLayer](#).

Examples:

[saslservtest.cpp](#), and [saslttest.cpp](#).

10.59.4.31 virtual int QCA::SASL::convertBytesWritten (qint64 *encryptedBytes*) [virtual]

Convert encrypted bytes written to plain text bytes written.

Implements [QCA::SecureLayer](#).

10.59.4.32 void QCA::SASL::serverStarted ()

This signal is emitted after the server has been successfully started.

10.59.4.33 void QCA::SASL::nextStep (const QByteArray & *stepData*)

This signal is emitted when there is data required to be sent over the network to complete the next step in the authentication process.

Parameters:

stepData the data to send over the network

10.59.4.34 void QCA::SASL::needParams (const [QCA::SASL::Params](#) & *params*)

This signal is emitted when the client needs additional parameters.

Set parameter values as necessary and then call [continueAfterParams\(\)](#).

10.59.4.35 void QCA::SASL::authCheck (const QString & *user*, const QString & *authzid*)

This signal is emitted when the server needs to perform the authentication check.

If the user and authzid are valid, call [continueAfterAuthCheck\(\)](#).

10.59.4.36 void QCA::SASL::authenticated ()

This signal is emitted when authentication is complete.

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

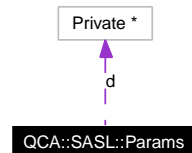
- [qca_securelayer.h](#)

10.60 QCA::SASL::Params Class Reference

Parameter flags for the [SASL](#) authentication.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::SASL::Params:



Public Member Functions

- [Params](#) (bool user, bool authzid, bool pass, bool realm)
- [Params](#) (const [Params](#) &from)
- [Params](#) & [operator=](#) (const [Params](#) &from)
- bool [needUsername](#) () const
- bool [canSendAuthzid](#) () const
- bool [needPassword](#) () const
- bool [canSendRealm](#) () const

10.60.1 Detailed Description

Parameter flags for the [SASL](#) authentication.

This is used to indicate which parameters are needed by [SASL](#) in order to complete the authentication process.

Examples:

[saslttest.cpp](#).

10.60.2 Constructor & Destructor Documentation

10.60.2.1 QCA::SASL::Params::Params (bool user, bool authzid, bool pass, bool realm)

Standard constructor.

The concept behind this is that you set each of the flags depending on which parameters are needed.

Parameters:

user the username is required

authzid the authorization identity is required

pass the password is required

realm the realm is required

10.60.2.2 QCA::SASL::Params::Params (const [Params](#) & *from*)

Standard copy constructor.

Parameters:

from the [Params](#) object to copy

10.60.3 Member Function Documentation

10.60.3.1 [Params](#)& QCA::SASL::Params::operator= (const [Params](#) & *from*)

Standard assignment operator.

Parameters:

from the [Params](#) object to assign from

10.60.3.2 bool QCA::SASL::Params::needUsername () const

User is needed.

Examples:

[sasltest.cpp](#).

10.60.3.3 bool QCA::SASL::Params::canSendAuthzid () const

An Authorization ID can be sent if desired.

Examples:

[sasltest.cpp](#).

10.60.3.4 bool QCA::SASL::Params::needPassword () const

Password is needed.

Examples:

[sasltest.cpp](#).

10.60.3.5 bool QCA::SASL::Params::canSendRealm () const

A Realm can be sent if desired.

Examples:

[sasltest.cpp](#).

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

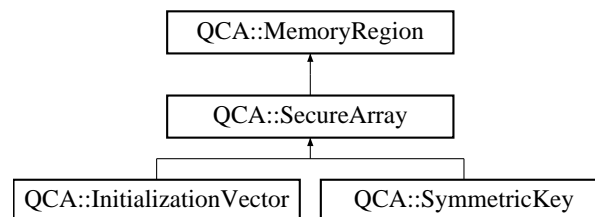
- [qca_securelayer.h](#)

10.61 QCA::SecureArray Class Reference

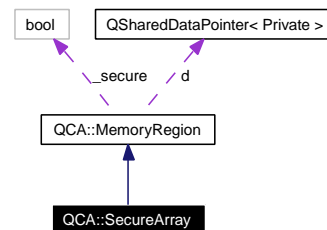
Secure array of bytes.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SecureArray::



Collaboration diagram for QCA::SecureArray:



Public Member Functions

- [SecureArray](#) ()
- [SecureArray](#) (int size, char ch=0)
- [SecureArray](#) (const char *str)
- [SecureArray](#) (const **QByteArray** &a)
- [SecureArray](#) (const [MemoryRegion](#) &a)
- [SecureArray](#) (const [SecureArray](#) &from)
- [SecureArray](#) & [operator=](#) (const [SecureArray](#) &from)
- [SecureArray](#) & [operator=](#) (const **QByteArray** &a)
- void [clear](#) ()
- char & [operator\[\]](#) (int index)
- const char & [operator\[\]](#) (int index) const
- char * [data](#) ()
- const char * [data](#) () const
- const char * [constData](#) () const
- char & [at](#) (int index)
- const char & [at](#) (int index) const
- int [size](#) () const
- bool [isEmpty](#) () const
- bool [resize](#) (int size)
- void [fill](#) (char fillChar, int fillToPosition=-1)
- **QByteArray** [toByteArray](#) () const

- [SecureArray](#) & [append](#) (const [SecureArray](#) &a)
- bool [operator==](#) (const [MemoryRegion](#) &other) const
- bool [operator!=](#) (const [MemoryRegion](#) &other) const
- [SecureArray](#) & [operator+=](#) (const [SecureArray](#) &a)

Protected Member Functions

- void [set](#) (const [SecureArray](#) &from)
- void [set](#) (const [QByteArray](#) &from)

10.61.1 Detailed Description

Secure array of bytes.

The [SecureArray](#) provides an array of memory from a pool that is, at least partly, secure. In this sense, secure means that the contents of the memory should not be made available to other applications. By comparison, a [QMemArray](#) (or subclass such as [QCString](#) or [QByteArray](#)) may be held in pages that might be swapped to disk or free'd without being cleared first.

Note that this class is implicitly shared (that is, copy on write).

Examples:

[aes-cmac.cpp](#).

10.61.2 Constructor & Destructor Documentation

10.61.2.1 QCA::SecureArray::SecureArray ()

Construct a secure byte array, zero length.

10.61.2.2 QCA::SecureArray::SecureArray (int *size*, char *ch* = 0) [explicit]

Construct a secure byte array of the specified length.

Parameters:

- size* the number of bytes in the array
- ch* the value every byte should be set to

10.61.2.3 QCA::SecureArray::SecureArray (const char * *str*)

Construct a secure byte array from a string.

Note that this copies, rather than references the source array

10.61.2.4 QCA::SecureArray::SecureArray (const [QByteArray](#) & *a*)

Construct a secure byte array from a [QByteArray](#).

Note that this copies, rather than references the source array

See also:

[operator=\(\)](#)

10.61.2.5 QCA::SecureArray::SecureArray (const [MemoryRegion](#) & *a*)

Construct a secure byte array from a [MemoryRegion](#).

Note that this copies, rather than references the source array

See also:

[operator=\(\)](#)

10.61.2.6 QCA::SecureArray::SecureArray (const [SecureArray](#) & *from*)

Construct a (shallow) copy of another secure byte array.

Parameters:

from the source of the data and length.

10.61.3 Member Function Documentation

10.61.3.1 [SecureArray&](#) QCA::SecureArray::operator= (const [SecureArray](#) & *from*)

Creates a reference, rather than a deep copy.

10.61.3.2 [SecureArray&](#) QCA::SecureArray::operator= (const QByteArray & *a*)

Creates a copy, rather than references.

Parameters:

a the array to copy from

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.3 void QCA::SecureArray::clear ()

Clears the contents of the array and makes it empty.

Examples:

[md5crypt.cpp](#).

10.61.3.4]

char& QCA::SecureArray::operator[] (int *index*)

Returns a reference to the byte at the index position.

Parameters:

index the zero-based offset to obtain

10.61.3.5]

const char& QCA::SecureArray::operator[] (int *index*) const

Returns a reference to the byte at the index position.

Parameters:

index the zero-based offset to obtain

10.61.3.6 char* QCA::SecureArray::data ()

Pointer to the data in the secure array.

You can use this for memcpy and similar functions. If you are trying to obtain data at a particular offset, you might be better off using [at\(\)](#) or [operator\[\]](#)

Reimplemented from [QCA::MemoryRegion](#).

Examples:

[ciphertest.cpp](#), [cmsexample.cpp](#), [hashtest.cpp](#), [mactest.cpp](#), [md5crypt.cpp](#), [publickeyexample.cpp](#), and [rsatest.cpp](#).

10.61.3.7 const char* QCA::SecureArray::data () const

Pointer to the data in the secure array.

You can use this for memcpy and similar functions. If you are trying to obtain data at a particular offset, you might be better off using [at\(\)](#) or [operator\[\]](#)

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.8 const char* QCA::SecureArray::constData () const

Pointer to the data in the secure array.

You can use this for memcpy and similar functions. If you are trying to obtain data at a particular offset, you might be better off using [at\(\)](#) or [operator\[\]](#)

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.9 char& QCA::SecureArray::at (int *index*)

Returns a reference to the byte at the index position.

Parameters:

index the zero-based offset to obtain

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.10 const char& QCA::SecureArray::at (int *index*) const

Returns a reference to the byte at the index position.

Parameters:

index the zero-based offset to obtain

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.11 int QCA::SecureArray::size () const

Returns the number of bytes in the array.

Reimplemented from [QCA::MemoryRegion](#).

Examples:

[aes-cmac.cpp](#), and [md5crypt.cpp](#).

10.61.3.12 bool QCA::SecureArray::isEmpty () const

Test if the array contains any bytes.

This is equivalent to testing (`size() != 0`). Note that if the array is allocated, `isEmpty()` is false (even if no data has been added)

Returns:

true if the array has zero length, otherwise false

Reimplemented from [QCA::MemoryRegion](#).

Examples:

[rsatest.cpp](#).

10.61.3.13 bool QCA::SecureArray::resize (int size)

Change the length of this array. If the new length is less than the old length, the extra information is (safely) discarded.

If the new length is equal to or greater than the old length, the existing data is copied into the array.

Parameters:

size the new length

Reimplemented from [QCA::MemoryRegion](#).

10.61.3.14 void QCA::SecureArray::fill (char fillChar, int fillToPosition = -1)

Fill the data array with a specified character.

Parameters:

fillChar the character to use as the fill

fillToPosition the number of characters to fill to. If not specified (or -1), fills array to current length.

Note:

This function does not extend the array - if you ask for fill beyond the current length, only the current length will be used.

The number of characters is 1 based, so if you ask for fill('x', 10), it will fill from

Examples:

[md5crypt.cpp](#).

10.61.3.15 QByteArray QCA::SecureArray::toByteArray () const

Copy the contents of the secure array out to a standard **QByteArray**.

Note that this performs a deep copy of the data.

Reimplemented from [QCA::MemoryRegion](#).

Examples:

[ciphertest.cpp](#), [hashtest.cpp](#), [mactest.cpp](#), [md5crypt.cpp](#), and [rsatest.cpp](#).

10.61.3.16 SecureArray& QCA::SecureArray::append (const SecureArray & a)

Append a secure byte array to the end of this array.

Examples:

[ciphertest.cpp](#), and [md5crypt.cpp](#).

10.61.3.17 bool QCA::SecureArray::operator== (const MemoryRegion & other) const

Equality operator.

Returns true if both arrays have the same data (and the same length, of course).

**10.61.3.18 bool QCA::SecureArray::operator!= (const MemoryRegion & other) const
[inline]**

Inequality operator.

Returns true if both arrays have different length, or the same length but different data.

10.61.3.19 SecureArray& QCA::SecureArray::operator+= (const SecureArray & a)

Append a secure byte array to the end of this array.

10.61.3.20 void QCA::SecureArray::set (const SecureArray & from) [protected]

Assign the contents of a provided byte array to this object.

Parameters:

from the byte array to copy

10.61.3.21 void QCA::SecureArray::set (const QByteArray &*from*) [protected]

Assign the contents of a provided byte array to this object.

Parameters:

from the byte array to copy

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

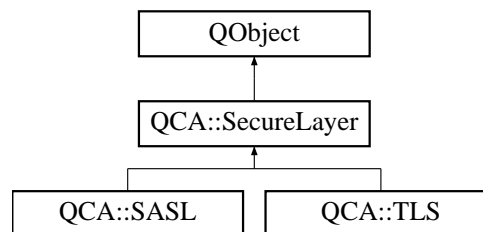
- [qca_tools.h](#)

10.62 QCA::SecureLayer Class Reference

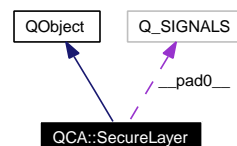
Abstract interface to a security layer.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SecureLayer::



Collaboration diagram for QCA::SecureLayer:



Public Member Functions

- [SecureLayer](#) ([QObject](#) *parent=0)
- virtual bool [isClosable](#) () const
- virtual int [bytesAvailable](#) () const =0
- virtual int [bytesOutgoingAvailable](#) () const =0
- virtual void [close](#) ()
- virtual void [write](#) (const [QByteArray](#) &a)=0
- virtual [QByteArray](#) [read](#) ()=0
- virtual void [writeIncoming](#) (const [QByteArray](#) &a)=0
- virtual [QByteArray](#) [readOutgoing](#) (int *plainBytes=0)=0
- virtual [QByteArray](#) [readUnprocessed](#) ()
- virtual int [convertBytesWritten](#) (qint64 encryptedBytes)=0
- void [readyReadOutgoing](#) ()
- void [closed](#) ()
- void [error](#) ()

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void readyRead()

10.62.1 Detailed Description

Abstract interface to a security layer.

[SecureLayer](#) is normally used between an application and a potentially insecure network. It provides secure communications over that network.

The concept is that (after some initial setup), the application can [write\(\)](#) some data to the [SecureLayer](#) implementation, and that data is encrypted (or otherwise protected, depending on the setup). The [SecureLayer](#) implementation then emits the [readyReadOutgoing\(\)](#) signal, and the application uses [readOutgoing\(\)](#) to retrieve the encrypted data from the [SecureLayer](#) implementation. The encrypted data is then sent out on the network.

When some encrypted data comes back from the network, the application does a [writeIncoming\(\)](#) to the [SecureLayer](#) implementation. Some time later, the [SecureLayer](#) implementation may emit [readyRead\(\)](#) to the application, which then [read\(\)](#)s the decrypted data from the [SecureLayer](#) implementation.

Note that sometimes data is sent or received between the [SecureLayer](#) implementation and the network without any data being sent between the application and the [SecureLayer](#) implementation. This is a result of the initial negotiation activities (which require network traffic to agree a configuration to use) and other overheads associated with the secure link.

10.62.2 Constructor & Destructor Documentation

10.62.2.1 QCA::SecureLayer::SecureLayer (QObject *parent = 0)

Constructor for an abstract secure communications layer.

Parameters:

parent the parent object for this object

10.62.3 Member Function Documentation

10.62.3.1 virtual bool QCA::SecureLayer::isClosable () const [virtual]

Returns true if the layer has a meaningful "close".

Reimplemented in [QCA::TLS](#).

10.62.3.2 virtual int QCA::SecureLayer::bytesAvailable () const [pure virtual]

Returns the number of bytes available to be [read\(\)](#) on the application side.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.3 virtual int QCA::SecureLayer::bytesOutgoingAvailable () const [pure virtual]

Returns the number of bytes available to be [readOutgoing\(\)](#) on the network side.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.4 virtual void QCA::SecureLayer::close () [virtual]

Close the link.

Note that this may not be meaningful / possible for all implementations.

See also:

[isClosable\(\)](#) for a test that verifies if the link can be closed.

Reimplemented in [QCA::TLS](#).

10.62.3.5 virtual void QCA::SecureLayer::write (const QByteArray & a) [pure virtual]

This method writes unencrypted (plain) data to the [SecureLayer](#) implementation.

You normally call this function on the application side.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.6 virtual QByteArray QCA::SecureLayer::read () [pure virtual]

This method reads decrypted (plain) data from the [SecureLayer](#) implementation.

You normally call this function on the application side after receiving the [readyRead\(\)](#) signal.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.7 virtual void QCA::SecureLayer::writeIncoming (const QByteArray & a) [pure virtual]

This method accepts encoded (typically encrypted) data for processing.

You normally call this function using data read from the network socket (e.g. using [QTcpSocket::readAll\(\)](#)) after receiving a signal that indicates that the socket has data to read.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.8 virtual QByteArray QCA::SecureLayer::readOutgoing (int * plainBytes = 0) [pure virtual]

This method provides encoded (typically encrypted) data.

You normally call this function to get data to write out to the network socket (e.g. using [QTcpSocket::write\(\)](#)) after receiving the [readyReadOutgoing\(\)](#) signal.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.9 virtual QByteArray QCA::SecureLayer::readUnprocessed () [virtual]

This allows you to read data without having it decrypted first.

This is intended to be used for protocols that close off the connection and return to plain text transfer. You do not normally need to use this function.

Reimplemented in [QCA::TLS](#).

10.62.3.10 `virtual int QCA::SecureLayer::convertBytesWritten (qint64 encryptedBytes)` [pure virtual]

Convert encrypted bytes written to plain text bytes written.

Implemented in [QCA::TLS](#), and [QCA::SASL](#).

10.62.3.11 `void QCA::SecureLayer::readyReadOutgoing ()`

This signal is emitted when [SecureLayer](#) has encrypted (network side) data ready to be read.

Typically you will connect this signal to a slot that reads the data (using [readOutgoing\(\)](#)) and writes it to a network socket.

10.62.3.12 `void QCA::SecureLayer::closed ()`

This signal is emitted when the [SecureLayer](#) connection is closed.

10.62.3.13 `void QCA::SecureLayer::error ()`

This signal is emitted when an error is detected.

You can determine the error type using `errorCode()`.

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

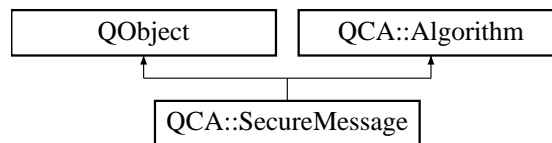
- [qca_securelayer.h](#)

10.63 QCA::SecureMessage Class Reference

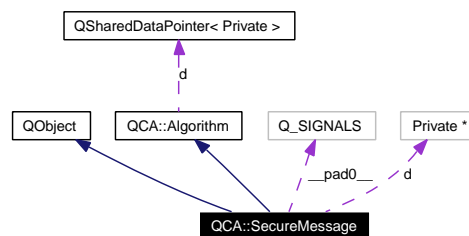
Class representing a secure message.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SecureMessage::



Collaboration diagram for QCA::SecureMessage:



Public Types

- enum [Type](#) { [OpenPGP](#), [CMS](#) }
- enum [SignMode](#) { [Message](#), [Clearsign](#), [Detached](#) }
- enum [Format](#) { [Binary](#), [Ascii](#) }
- enum [Error](#) {
[ErrorPassphrase](#), [ErrorFormat](#), [ErrorSignerExpired](#), [ErrorSignerInvalid](#),
[ErrorEncryptExpired](#), [ErrorEncryptUntrusted](#), [ErrorEncryptInvalid](#), [ErrorNeedCard](#),
[ErrorCertKeyMismatch](#), [ErrorUnknown](#) }

Public Member Functions

- [SecureMessage](#) ([SecureMessageSystem](#) *system)
- [Type](#) type () const
- bool [canSignMultiple](#) () const
- bool [canClearsign](#) () const
- bool [canSignAndEncrypt](#) () const
- void [reset](#) ()
- bool [bundleSignerEnabled](#) () const
- bool [smimeAttributesEnabled](#) () const
- [Format](#) format () const
- [SecureMessageKeyList](#) [recipientKeys](#) () const
- [SecureMessageKeyList](#) [signerKeys](#) () const
- void [setBundleSignerEnabled](#) (bool b)

- void [setSMIMEAttributesEnabled](#) (bool b)
- void [setFormat](#) ([Format](#) f)
- void [setRecipient](#) (const [SecureMessageKey](#) &key)
- void [setRecipients](#) (const [SecureMessageKeyList](#) &keys)
- void [setSigner](#) (const [SecureMessageKey](#) &key)
- void [setSigners](#) (const [SecureMessageKeyList](#) &keys)
- void [startEncrypt](#) ()
- void [startDecrypt](#) ()
- void [startSign](#) ([SignMode](#) m=Message)
- void [startVerify](#) (const [QByteArray](#) &detachedSig=[QByteArray](#)())
- void [startSignAndEncrypt](#) ()
- void [update](#) (const [QByteArray](#) &in)
- [QByteArray](#) [read](#) ()
- int [bytesAvailable](#) () const
- void [end](#) ()
- bool [waitForFinished](#) (int msec=30000)
- bool [success](#) () const
- [Error](#) [errorCode](#) () const
- [QByteArray](#) [signature](#) () const
- [QString](#) [hashName](#) () const
- bool [wasSigned](#) () const
- bool [verifySuccess](#) () const
- [SecureMessageSignature](#) [signer](#) () const
- [SecureMessageSignatureList](#) [signers](#) () const
- [QString](#) [diagnosticText](#) () const
- void [bytesWritten](#) (int bytes)
- void [finished](#) ()

Public Attributes

- Q_SIGNALS [__pad0__](#): void readyRead()

Friends

- class [Private](#)

10.63.1 Detailed Description

Class representing a secure message.

[SecureMessage](#) presents a unified interface for working with both [OpenPGP](#) and [CMS](#) (S/MIME) messages. Prepare the object by calling [setFormat\(\)](#), [setRecipient\(\)](#), and [setSigner\(\)](#) as necessary, and then begin the operation by calling an appropriate 'start' function, such as [startSign\(\)](#).

Here is an example of how to perform a Clearsign operation using PGP:

```
// first make the SecureMessageKey
PGPKey myPGPKey = getSecretKeyFromSomewhere();
SecureMessageKey key;
key.setPGPSecretKey(myPGPKey);
```

```
// our data to sign
QByteArray plain = "Hello, world";

// let's do it
OpenPGP pgp;
SecureMessage msg(&pgp);
msg.setSigner(key);
msg.startSign(SecureMessage::Clearsign);
msg.update(plain);
msg.end();
msg.waitForFinished(-1);

if(msg.success())
{
    QByteArray result = msg.read();
    // result now contains the clearsign text data
}
else
{
    // error
    ...
}
```

Performing a [CMS](#) sign operation is similar. Simply set up the [SecureMessageKey](#) with a [Certificate](#) instead of a [PGPKey](#), and operate on a [CMS](#) object instead of an [OpenPGP](#) object.

See also:

[SecureMessageKey](#)
[SecureMessageSignature](#)
[OpenPGP](#)
[CMS](#)

Examples:

[cmsexample.cpp](#), and [publickeyexample.cpp](#).

10.63.2 Member Enumeration Documentation

10.63.2.1 enum [QCA::SecureMessage::Type](#)

The type of secure message.

Enumerator:

OpenPGP a Pretty Good Privacy message
CMS a Cryptographic Message Syntax message

10.63.2.2 enum [QCA::SecureMessage::SignMode](#)

The type of message signature.

Enumerator:

Message the message includes the signature
Clearsign the message is clear signed
Detached the signature is detached

10.63.2.3 enum [QCA::SecureMessage::Format](#)

Formats for secure messages.

Enumerator:

Binary DER/binary.

Ascii PEM/ascii-armored.

10.63.2.4 enum [QCA::SecureMessage::Error](#)

Errors for secure messages.

Enumerator:

ErrorPassphrase passphrase was either wrong or not provided

ErrorFormat input format was bad

ErrorSignerExpired signing key is expired

ErrorSignerInvalid signing key is invalid in some way

ErrorEncryptExpired encrypting key is expired

ErrorEncryptUntrusted encrypting key is untrusted

ErrorEncryptInvalid encrypting key is invalid in some way

ErrorNeedCard pgp card is missing

ErrorCertKeyMismatch certificate and private key don't match

ErrorUnknown other error

10.63.3 Constructor & Destructor Documentation

10.63.3.1 [QCA::SecureMessage::SecureMessage](#) ([SecureMessageSystem](#) * *system*)

Create a new secure message.

This constructor uses an existing [SecureMessageSystem](#) object (for example, an [OpenPGP](#) or [CMS](#) object) to generate a specific kind of secure message.

Parameters:

system a pre-existing and configured [SecureMessageSystem](#) object

10.63.4 Member Function Documentation

10.63.4.1 **Type** [QCA::SecureMessage::type](#) () const

The Type of secure message.

Reimplemented from [QCA::Algorithm](#).

10.63.4.2 bool QCA::SecureMessage::canSignMultiple () const

Test if the message type supports multiple (parallel) signatures.

Returns:

true if the secure message support multiple parallel signatures

Note:

PGP cannot do this - it is primarily a CMS feature

10.63.4.3 bool QCA::SecureMessage::canClearsign () const

True if the SecureMessageSystem can clearsign messages.

Note:

CMS cannot clearsign - this is normally only available for PGP

10.63.4.4 bool QCA::SecureMessage::canSignAndEncrypt () const

True if the SecureMessageSystem can both sign and encrypt (in the same operation).

Note:

CMS cannot do an integrated sign/encrypt - this is normally only available for PGP. You can do separate signing and encrypting operations on the same message with CMS though.

10.63.4.5 void QCA::SecureMessage::reset ()

Reset the object state to that of original construction.

Now a new operation can be performed immediately.

10.63.4.6 bool QCA::SecureMessage::bundleSignerEnabled () const

Returns true if bundling of the signer certificate chain is enabled.

10.63.4.7 bool QCA::SecureMessage::smimeAttributesEnabled () const

Returns true if inclusion of S/MIME attributes is enabled.

10.63.4.8 Format QCA::SecureMessage::format () const

Return the format type set for this message.

10.63.4.9 SecureMessageKeyList QCA::SecureMessage::recipientKeys () const

Return the recipient(s) set for this message with setRecipient() or setRecipients().

10.63.4.10 SecureMessageKeyList QCA::SecureMessage::signerKeys () const

Return the signer(s) set for this message with [setSigner\(\)](#) or [setSigners\(\)](#).

10.63.4.11 void QCA::SecureMessage::setBundleSignerEnabled (bool *b*)

For [CMS](#) only, this will bundle the signer certificate chain into the message.

This allows a message to be verified on its own, without the need to have obtained the signer's certificate in advance. Email clients using S/MIME often bundle the signer, greatly simplifying key management.

This behavior is enabled by default.

10.63.4.12 void QCA::SecureMessage::setSMIMEAttributesEnabled (bool *b*)

For [CMS](#) only, this will put extra attributes into the message related to S/MIME, such as the preferred type of algorithm to use in replies.

The attributes used are decided by the provider.

This behavior is enabled by default.

10.63.4.13 void QCA::SecureMessage::setFormat ([Format](#) *f*)

Set the Format used for messages.

The default is Binary.

Parameters:

f whether to use Binary or Ascii

10.63.4.14 void QCA::SecureMessage::setRecipient (const [SecureMessageKey](#) & *key*)

Set the recipient for an encrypted message.

See also:

[setRecipients](#)

10.63.4.15 void QCA::SecureMessage::setRecipients (const SecureMessageKeyList & *keys*)

Set the list of recipients for an encrypted message.

For a list with one item, this has the same effect as [setRecipient](#).

See also:

[setRecipient](#)

10.63.4.16 void QCA::SecureMessage::setSigner (const SecureMessageKey & key)

Set the signer for a signed message.

This is used for both creating signed messages as well as for verifying CMS messages that have no signer bundled.

See also:

[setSigners](#)

10.63.4.17 void QCA::SecureMessage::setSigners (const SecureMessageKeyList & keys)

Set the list of signers for a signed message.

This is used for both creating signed messages as well as for verifying CMS messages that have no signer bundled.

For a list with one item, this has the same effect as setSigner.

See also:

[setSigner](#)

10.63.4.18 void QCA::SecureMessage::startEncrypt ()

Start an encryption operation.

You will normally use this with some code along these lines:

```
encryptingObj.startEncrypt();
encryptingObj.update(message);
// perhaps some more update()s
encryptingObj.end();
```

Each [update\(\)](#) may (or may not) result in some encrypted data, as indicated by the [readyRead\(\)](#) signal being emitted. Alternatively, you can wait until the whole message is available (using either [waitForFinished\(\)](#), or use the [finished\(\)](#) signal). The encrypted message can then be read using the [read\(\)](#) method.

10.63.4.19 void QCA::SecureMessage::startDecrypt ()

Start an decryption operation.

You will normally use this with some code along these lines:

```
decryptingObj.startEncrypt();
decryptingObj.update(message);
// perhaps some more update()s
decryptingObj.end();
```

Each [update\(\)](#) may (or may not) result in some decrypted data, as indicated by the [readyRead\(\)](#) signal being emitted. Alternatively, you can wait until the whole message is available (using either [waitForFinished\(\)](#), or the [finished\(\)](#) signal). The decrypted message can then be read using the [read\(\)](#) method.

Note:

If decrypted result is also signed (not for CMS), then the signature will be verified during this operation.

10.63.4.20 void QCA::SecureMessage::startSign (SignMode *m* = Message)

Start a signing operation.

You will normally use this with some code along these lines:

```
signingObj.startSign(QCA::SecureMessage::Detached)
signingObj.update(message);
// perhaps some more update()s
signingObj.end();
```

For Detached signatures, you won't get any results until the whole process is done - you either [waitForFinished\(\)](#), or use the [finished\(\)](#) signal, to figure out when you can get the signature (using the [signature\(\)](#) method, not using [read\(\)](#)). For other formats, you can use the [readyRead\(\)](#) signal to determine when there may be part of a signed message to [read\(\)](#).

Parameters:

m the mode that will be used to generate the signature

10.63.4.21 void QCA::SecureMessage::startVerify (const QByteArray & *detachedSig* = QByteArray ())

Start a verification operation.

Parameters:

detachedSig the detached signature to verify. Do not pass a signature for other signature types.

10.63.4.22 void QCA::SecureMessage::startSignAndEncrypt ()

Start a combined signing and encrypting operation.

You use this in the same way as [startEncrypt\(\)](#).

Note:

This may not be possible (e.g. [CMS](#) cannot do this) - see [canSignAndEncrypt\(\)](#) for a suitable test.

10.63.4.23 void QCA::SecureMessage::update (const QByteArray & *in*)

Process a message (or the next part of a message) in the current operation.

You need to have already set up the message ([startEncrypt\(\)](#), [startDecrypt\(\)](#), [startSign\(\)](#), [startSignAndEncrypt\(\)](#) and [startVerify\(\)](#)) before calling this method.

Parameters:

in the data to process

10.63.4.24 QByteArray QCA::SecureMessage::read ()

Read the available data.

Note:

For detached signatures, you don't get anything back using this method. Use [signature\(\)](#) to get the detached [signature\(\)](#).

10.63.4.25 int QCA::SecureMessage::bytesAvailable () const

The number of bytes available to be read.

10.63.4.26 void QCA::SecureMessage::end ()

Complete an operation.

You need to call this method after you have processed the message (which you pass in as the argument to [update\(\)](#)).

Note:

the results of the operation are not available as soon as this method returns. You need to wait for the [finished\(\)](#) signal, or use [waitForFinished\(\)](#).

10.63.4.27 bool QCA::SecureMessage::waitForFinished (int *msecs* = 30000)

Block until the operation (encryption, decryption, signing or verifying) completes.

Parameters:

msecs the number of milliseconds to wait for the operation to complete. Pass -1 to wait indefinitely.

Note:

You should not use this in GUI applications where the blocking behaviour looks like a hung application. Instead, connect the [finished\(\)](#) signal to a slot that handles the results. This synchronous operation may require event handling, and so it must not be called from the same thread as an [EventHandler](#).

10.63.4.28 bool QCA::SecureMessage::success () const

Indicates whether or not the operation was successful or failed.

If this function returns false, then the reason for failure can be obtained with [errorCode\(\)](#).

See also:

[errorCode](#)
[diagnosticText](#)

10.63.4.29 Error QCA::SecureMessage::errorCode () const

Returns the failure code.

See also:

[success](#)
[diagnosticText](#)

10.63.4.30 QByteArray QCA::SecureMessage::signature () const

The signature for the message.

This is only used for Detached signatures. For other message types, you get the message and signature together using [read\(\)](#).

10.63.4.31 QString QCA::SecureMessage::hashName () const

The name of the hash used for the signature process.

10.63.4.32 bool QCA::SecureMessage::wasSigned () const

Test if the message was signed.

This is true for [OpenPGP](#) if the decrypted message was also signed.

Returns:

true if the message was signed.

10.63.4.33 bool QCA::SecureMessage::verifySuccess () const

Verify that the message signature is correct.

Returns:

true if the signature is valid for the message, otherwise return false

10.63.4.34 SecureMessageSignature QCA::SecureMessage::signer () const

Information on the signer for the message.

10.63.4.35 SecureMessageSignatureList QCA::SecureMessage::signers () const

Information on the signers for the message.

This is only meaningful if the message type supports multiple signatures (see [canSignMultiple\(\)](#) for a suitable test).

10.63.4.36 QString QCA::SecureMessage::diagnosticText () const

Returns a log of technical information about the operation, which may be useful for presenting to the user in an advanced error dialog.

10.63.4.37 void QCA::SecureMessage::bytesWritten (int *bytes*)

This signal is emitted when data has been accepted by the message processor.

10.63.4.38 void QCA::SecureMessage::finished ()

This signal is emitted when the message is fully processed.

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

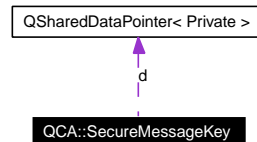
- [qca_securemessage.h](#)

10.64 QCA::SecureMessageKey Class Reference

Key for [SecureMessage](#) system.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::SecureMessageKey:



Public Types

- enum [Type](#) { [None](#), [PGP](#), [X509](#) }

Public Member Functions

- [SecureMessageKey](#) ()
- [SecureMessageKey](#) (const [SecureMessageKey](#) &from)
- [SecureMessageKey](#) & operator= (const [SecureMessageKey](#) &from)
- bool [isNull](#) () const
- [Type](#) [type](#) () const
- [PGPKey](#) [pgpPublicKey](#) () const
- [PGPKey](#) [pgpSecretKey](#) () const
- void [setPGPPublicKey](#) (const [PGPKey](#) &pub)
- void [setPGPSecretKey](#) (const [PGPKey](#) &sec)
- [CertificateChain](#) [x509CertificateChain](#) () const
- [PrivateKey](#) [x509PrivateKey](#) () const
- void [setX509CertificateChain](#) (const [CertificateChain](#) &c)
- void [setX509PrivateKey](#) (const [PrivateKey](#) &k)
- void [setX509KeyBundle](#) (const [KeyBundle](#) &kb)
- bool [havePrivate](#) () const
- [QString](#) [name](#) () const

10.64.1 Detailed Description

Key for [SecureMessage](#) system.

Examples:

[cmsexample.cpp](#), and [publickeyexample.cpp](#).

10.64.2 Member Enumeration Documentation

10.64.2.1 enum [QCA::SecureMessageKey::Type](#)

The key type.

Enumerator:

- None* no key
- PGP* Pretty Good Privacy key.
- X509* X.509 [CMS](#) key.

10.64.3 Constructor & Destructor Documentation**10.64.3.1 QCA::SecureMessageKey::SecureMessageKey ()**

Construct an empty key.

10.64.3.2 QCA::SecureMessageKey::SecureMessageKey (const [SecureMessageKey](#) & *from*)

Standard copy constructor.

Parameters:

from the source key

10.64.4 Member Function Documentation**10.64.4.1 [SecureMessageKey](#) & QCA::SecureMessageKey::operator= (const [SecureMessageKey](#) & *from*)**

Standard assignment operator.

Parameters:

from the source key

10.64.4.2 bool QCA::SecureMessageKey::isNull () const

Returns true for null object.

10.64.4.3 [Type](#) QCA::SecureMessageKey::type () const

The key type.

10.64.4.4 [PGPKey](#) QCA::SecureMessageKey::pgpPublicKey () const

Public key part of a PGP key.

10.64.4.5 [PGPKey](#) QCA::SecureMessageKey::pgpSecretKey () const

Private key part of a PGP key.

10.64.4.6 void QCA::SecureMessageKey::setPGPPublicKey (const [PGPKey](#) & *pub*)

Set the public key part of a PGP key.

Parameters:

pub the PGP public key

10.64.4.7 void QCA::SecureMessageKey::setPGPSecretKey (const [PGPKey](#) & *sec*)

Set the private key part of a PGP key.

Parameters:

sec the PGP secretkey

10.64.4.8 [CertificateChain](#) QCA::SecureMessageKey::x509CertificateChain () const

The X.509 certificate chain (public part) for this key.

10.64.4.9 [PrivateKey](#) QCA::SecureMessageKey::x509PrivateKey () const

The X.509 private key part of this key.

10.64.4.10 void QCA::SecureMessageKey::setX509CertificateChain (const [CertificateChain](#) & *c*)

Set the public key part of this X.509 key.

Examples:

[cmsexample.cpp](#), and [publickeyexample.cpp](#).

10.64.4.11 void QCA::SecureMessageKey::setX509PrivateKey (const [PrivateKey](#) & *k*)

Set the private key part of this X.509 key.

Examples:

[cmsexample.cpp](#).

10.64.4.12 void QCA::SecureMessageKey::setX509KeyBundle (const [KeyBundle](#) & *kb*)

Set the public and private part of this X.509 key with [KeyBundle](#).

10.64.4.13 bool QCA::SecureMessageKey::havePrivate () const

Test if this key contains a private key part.

10.64.4.14 QString QCA::SecureMessageKey::name () const

The name associated with this key.

For a PGP key, this is the primary user ID

For an X.509 key, this is the Common Name

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

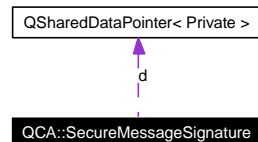
- [qca_securemessage.h](#)

10.65 QCA::SecureMessageSignature Class Reference

[SecureMessage](#) signature.

```
#include <QtCrypto>
```

Collaboration diagram for QCA::SecureMessageSignature:



Public Types

- enum [IdentityResult](#) { [Valid](#), [InvalidSignature](#), [InvalidKey](#), [NoKey](#) }

Public Member Functions

- [SecureMessageSignature](#) ()
- [SecureMessageSignature](#) ([IdentityResult](#) r, [Validity](#) v, const [SecureMessageKey](#) &key, const [QDateTime](#) &ts)
- [SecureMessageSignature](#) (const [SecureMessageSignature](#) &from)
- [SecureMessageSignature](#) & operator= (const [SecureMessageSignature](#) &from)
- [IdentityResult](#) [identityResult](#) () const
- [Validity](#) [keyValidity](#) () const
- [SecureMessageKey](#) [key](#) () const
- [QDateTime](#) [timestamp](#) () const

10.65.1 Detailed Description

[SecureMessage](#) signature.

Examples:

[cmsexample.cpp](#).

10.65.2 Member Enumeration Documentation

10.65.2.1 enum [QCA::SecureMessageSignature::IdentityResult](#)

The result of identity verification.

Enumerator:

- Valid*** identity is verified, matches signature
- InvalidSignature*** valid key provided, but signature failed
- InvalidKey*** invalid key provided
- NoKey*** identity unknown

10.65.3 Constructor & Destructor Documentation

10.65.3.1 QCA::SecureMessageSignature::SecureMessageSignature ()

Create an empty signature check object.

10.65.3.2 QCA::SecureMessageSignature::SecureMessageSignature ([IdentityResult](#) *r*, [Validity](#) *v*, const [SecureMessageKey](#) & *key*, const QDateTime & *ts*)

Create a signature check object.

10.65.3.3 QCA::SecureMessageSignature::SecureMessageSignature (const [SecureMessageSignature](#) & *from*)

Standard copy constructor.

Parameters:

from the source signature object

10.65.4 Member Function Documentation

10.65.4.1 [SecureMessageSignature&](#) QCA::SecureMessageSignature::operator= (const [SecureMessageSignature](#) & *from*)

Standard assignment operator.

Parameters:

from the source signature object

10.65.4.2 [IdentityResult](#) QCA::SecureMessageSignature::identityResult () const

get the results of the identity check on this signature

10.65.4.3 [Validity](#) QCA::SecureMessageSignature::keyValidity () const

get the results of the key validation check on this signature

10.65.4.4 [SecureMessageKey](#) QCA::SecureMessageSignature::key () const

get the key associated with this signature

10.65.4.5 QDateTime QCA::SecureMessageSignature::timestamp () const

get the timestamp associated with this signature

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

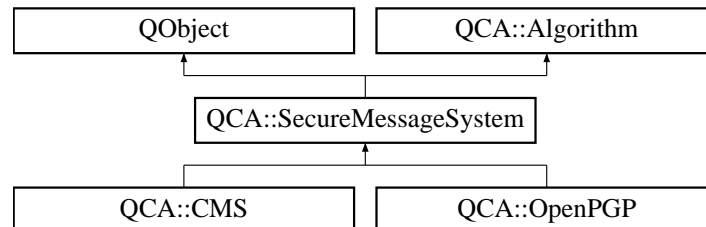
- [qca_securemessage.h](#)

10.66 QCA::SecureMessageSystem Class Reference

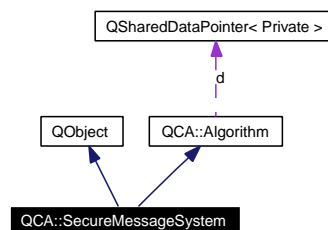
Abstract superclass for secure messaging systems.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SecureMessageSystem::



Collaboration diagram for QCA::SecureMessageSystem:



Protected Member Functions

- [SecureMessageSystem](#) (`QObject *parent`, `const QString &type`, `const QString &provider`)

10.66.1 Detailed Description

Abstract superclass for secure messaging systems.

See also:

[SecureMessage](#)
[SecureMessageKey](#)

10.66.2 Constructor & Destructor Documentation

10.66.2.1 QCA::SecureMessageSystem::SecureMessageSystem (`QObject *parent`, `const QString &type`, `const QString &provider`) [protected]

Protected constructor for [SecureMessageSystem](#) classes.

You are meant to be using a subclass (such as [OpenPGP](#) or [CMS](#)) - you only need to worry about this class if you are creating a whole new [SecureMessageSystem](#) type.

Parameters:

parent the parent object for this object

type the name of the Type of [SecureMessageSystem](#) to create
provider the provider to use, if a specific provider is required.

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

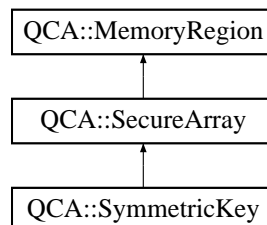
- [qca_securemessage.h](#)

10.67 QCA::SymmetricKey Class Reference

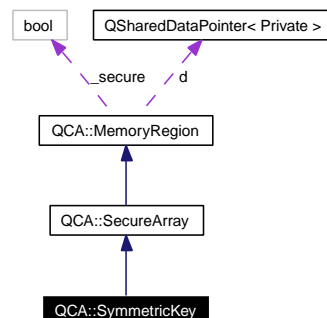
Container for keys for symmetric encryption algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::SymmetricKey::



Collaboration diagram for QCA::SymmetricKey:



Public Member Functions

- [SymmetricKey \(\)](#)
- [SymmetricKey \(int size\)](#)
- [SymmetricKey \(const SecureArray &a\)](#)
- [SymmetricKey \(const QByteArray &a\)](#)
- [bool isWeakDESKey \(\)](#)

10.67.1 Detailed Description

Container for keys for symmetric encryption algorithms.

Examples:

[aes-cmac.cpp](#), [ciphertest.cpp](#), and [mactest.cpp](#).

10.67.2 Constructor & Destructor Documentation

10.67.2.1 QCA::SymmetricKey::SymmetricKey ()

Construct an empty (zero length) key.

10.67.2.2 QCA::SymmetricKey::SymmetricKey (int *size*)

Construct an key of specified size, with random contents.

This is intended to be used as a random session key.

Parameters:

size the number of bytes for the key

10.67.2.3 QCA::SymmetricKey::SymmetricKey (const SecureArray & *a*)

Construct a key from a provided byte array.

Parameters:

a the byte array to copy

10.67.2.4 QCA::SymmetricKey::SymmetricKey (const QByteArray & *a*)

Construct a key from a provided byte array.

Parameters:

a the byte array to copy

10.67.3 Member Function Documentation

10.67.3.1 bool QCA::SymmetricKey::isWeakDESKey ()

Test for weak DES keys.

Returns:

true if the key is a weak key for DES

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

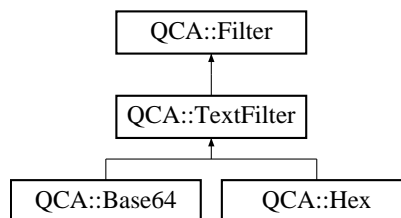
- [qca_core.h](#)

10.68 QCA::TextFilter Class Reference

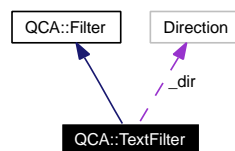
Superclass for text based filtering algorithms.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::TextFilter:



Collaboration diagram for QCA::TextFilter:



Public Member Functions

- [TextFilter](#) ([Direction](#) dir)
- void [setup](#) ([Direction](#) dir)
- [Direction](#) [direction](#) () const
- [MemoryRegion](#) [encode](#) (const [MemoryRegion](#) &a)
- [MemoryRegion](#) [decode](#) (const [MemoryRegion](#) &a)
- [QString](#) [arrayToString](#) (const [MemoryRegion](#) &a)
- [MemoryRegion](#) [stringToArray](#) (const [QString](#) &s)
- [QString](#) [encodeString](#) (const [QString](#) &s)
- [QString](#) [decodeString](#) (const [QString](#) &s)

Protected Attributes

- [Direction](#) _dir

10.68.1 Detailed Description

Superclass for text based filtering algorithms.

This differs from [Filter](#) in that it has the concept of an algorithm that works in two directions, and supports operations on [QString](#) arguments.

10.68.2 Constructor & Destructor Documentation

10.68.2.1 QCA::TextFilter::TextFilter ([Direction](#) *dir*)

Standard constructor.

Parameters:

dir the [Direction](#) that this [TextFilter](#) should use.

10.68.3 Member Function Documentation

10.68.3.1 void QCA::TextFilter::setup ([Direction](#) *dir*)

Reset the [TextFilter](#).

Parameters:

dir the [Direction](#) that this [TextFilter](#) should use.

10.68.3.2 [Direction](#) QCA::TextFilter::direction () const

The direction the [TextFilter](#) is set up to use.

10.68.3.3 [MemoryRegion](#) QCA::TextFilter::encode (const [MemoryRegion](#) & *a*)

Process an array in the "forward" direction, returning an array.

This method runs in the forward direction, so for something like a [Base64](#) encoding, it takes the "native" array, and returns that array encoded in base64.

Parameters:

a the array to encode

10.68.3.4 [MemoryRegion](#) QCA::TextFilter::decode (const [MemoryRegion](#) & *a*)

Process an array in the "reverse" direction, returning an array.

This method runs in the reverse direction, so for something like a [Base64](#) encoding, it takes a [Base64](#) encoded array, and returns the "native" representation.

Parameters:

a the array to decode

10.68.3.5 QString QCA::TextFilter::arrayToString (const [MemoryRegion](#) & *a*)

Process an array in the "forward" direction, returning a [QString](#).

This is equivalent to [encode\(\)](#), except that it returns a [QString](#), rather than a byte array.

Parameters:

a the array to encode

Examples:

[base64test.cpp](#), [cmsexample.cpp](#), [hextest.cpp](#), [publickeyexample.cpp](#), [saslservtest.cpp](#), and [saslttest.cpp](#).

10.68.3.6 [MemoryRegion](#) `QCA::TextFilter::stringToArray (const QString & s)`

Process an string in the "reverse" direction, returning a byte array.

This is equivalent to [decode\(\)](#), except that it takes a **QString**, rather than a byte array.

Parameters:

s the array to decode

10.68.3.7 `QString QCA::TextFilter::encodeString (const QString & s)`

Process a string in the "forward" direction, returning a string.

This is equivalent to [encode\(\)](#), except that it takes and returns a **QString**, rather than byte arrays.

Parameters:

s the string to encode

10.68.3.8 `QString QCA::TextFilter::decodeString (const QString & s)`

Process a string in the "reverse" direction, returning a string.

This is equivalent to [decode\(\)](#), except that it takes and returns a **QString**, rather than byte arrays.

Parameters:

s the string to decode

Examples:

[base64test.cpp](#), and [hextest.cpp](#).

10.68.4 Member Data Documentation**10.68.4.1 [Direction](#) `QCA::TextFilter::_dir` [protected]**

Internal state variable for the Direction that the filter operates in.

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

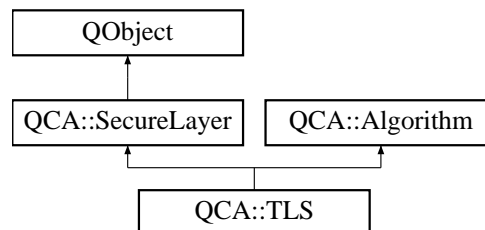
- [qca_textfilter.h](#)

10.69 QCA::TLS Class Reference

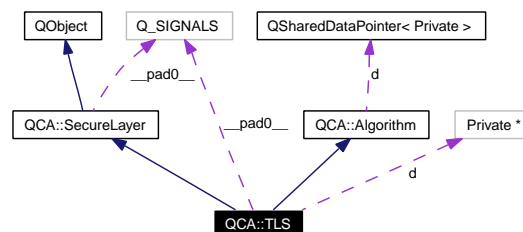
Transport Layer Security / Secure Socket Layer.

```
#include <QtCrypto>
```

Inheritance diagram for QCA::TLS::



Collaboration diagram for QCA::TLS:



Public Types

- enum [Mode](#) { [Stream](#), [Datagram](#) }
- enum [Version](#) { [TLS_v1](#), [SSL_v3](#), [SSL_v2](#), [DTLS_v1](#) }
- enum [Error](#) {
[ErrorSignerExpired](#), [ErrorSignerInvalid](#), [ErrorCertKeyMismatch](#), [ErrorInit](#),
[ErrorHandshake](#), [ErrorCrypt](#) }
- enum [IdentityResult](#) { [Valid](#), [HostMismatch](#), [InvalidCertificate](#), [NoCertificate](#) }

Public Member Functions

- [TLS](#) ([QObject](#) *parent=0, const [QString](#) &provider=[QString](#)())
- [TLS](#) ([Mode](#) mode, [QObject](#) *parent=0, const [QString](#) &provider=[QString](#)())
- [~TLS](#) ()
- void [reset](#) ()
- [QStringList](#) [supportedCipherSuites](#) (const [Version](#) &version=[TLS_v1](#)) const
- void [setCertificate](#) (const [CertificateChain](#) &cert, const [PrivateKey](#) &key)
- void [setCertificate](#) (const [KeyBundle](#) &kb)
- [CertificateCollection](#) [trustedCertificates](#) () const
- void [setTrustedCertificates](#) (const [CertificateCollection](#) &trusted)
- void [setConstraints](#) ([SecurityLevel](#) s)
- void [setConstraints](#) (int minSSF, int maxSSF)

- void [setConstraints](#) (const **QStringList** &cipherSuiteList)
- **QList**< [CertificateInfoOrdered](#) > [issuerList](#) () const
- void [setIssuerList](#) (const **QList**< [CertificateInfoOrdered](#) > &issuers)
- void [setSession](#) (const [TLSSession](#) &session)
- bool [canCompress](#) () const
- bool [canSetHostName](#) () const
- bool [compressionEnabled](#) () const
- void [setCompressionEnabled](#) (bool b)
- **QString** [hostName](#) () const
- void [startClient](#) (const **QString** &host=**QString**())
- void [startServer](#) ()
- void [continueAfterStep](#) ()
- bool [isHandshaken](#) () const
- bool [isCompressed](#) () const
- [Version](#) [version](#) () const
- **QString** [cipherSuite](#) () const
- int [cipherBits](#) () const
- int [cipherMaxBits](#) () const
- [TLSSession](#) [session](#) () const
- [Error](#) [errorCode](#) () const
- [IdentityResult](#) [peerIdentityResult](#) () const
- [Validity](#) [peerCertificateValidity](#) () const
- [CertificateChain](#) [localCertificateChain](#) () const
- [PrivateKey](#) [localPrivateKey](#) () const
- [CertificateChain](#) [peerCertificateChain](#) () const
- virtual bool [isClosable](#) () const
- virtual int [bytesAvailable](#) () const
- virtual int [bytesOutgoingAvailable](#) () const
- virtual void [close](#) ()
- virtual void [write](#) (const **QByteArray** &a)
- virtual **QByteArray** [read](#) ()
- virtual void [writeIncoming](#) (const **QByteArray** &a)
- virtual **QByteArray** [readOutgoing](#) (int *plainBytes=0)
- virtual **QByteArray** [readUnprocessed](#) ()
- virtual int [convertBytesWritten](#) (qint64 encryptedBytes)
- int [packetsAvailable](#) () const
- int [packetsOutgoingAvailable](#) () const
- int [packetMTU](#) () const
- void [setPacketMTU](#) (int size) const
- void [certificateRequested](#) ()
- void [peerCertificateAvailable](#) ()
- void [handshaken](#) ()

Public Attributes

- Q_SIGNALS [__pad0__](#): void [hostNameReceived](#)()

Protected Member Functions

- void **connectNotify** (const char *signal)
- void **disconnectNotify** (const char *signal)

Friends

- class **Private**

10.69.1 Detailed Description

Transport Layer Security / Secure Socket Layer.

Transport Layer Security (TLS) is the current state-of-the-art in secure transport mechanisms over the internet. It can be used in a way where only one side of the link needs to authenticate to the other. This makes it very useful for servers to provide their identity to clients. Note that it is possible to use TLS to authenticate both client and server.

TLS is a IETF standard ([RFC2712](#) for [TLS](#) version 1.0) based on earlier Netscape work on Secure Socket Layer (SSL version 2 and SSL version 3). New applications should use at least [TLS](#) 1.0, and SSL version 2 should be avoided due to known security problems.

Examples:

[sslservtest.cpp](#), [ssltest.cpp](#), and [tlssocket.cpp](#).

10.69.2 Member Enumeration Documentation

10.69.2.1 enum [QCA::TLS::Mode](#)

Operating mode.

Enumerator:

Stream stream mode

Datagram datagram mode

10.69.2.2 enum [QCA::TLS::Version](#)

Version of TLS or SSL.

Enumerator:

TLS_v1 Transport Layer Security, version 1.

SSL_v3 Secure Socket Layer, version 3.

SSL_v2 Secure Socket Layer, version 2.

DTLS_v1 Datagram Transport Layer Security, version 1.

10.69.2.3 enum [QCA::TLS::Error](#)

Type of error.

Enumerator:

- ErrorSignerExpired* local certificate is expired
- ErrorSignerInvalid* local certificate is invalid in some way
- ErrorCertKeyMismatch* certificate and private key don't match
- ErrorInit* problem starting up TLS
- ErrorHandshake* problem during the negotiation
- ErrorCrypt* problem at anytime after

10.69.2.4 enum [QCA::TLS::IdentityResult](#)

Type of identity.

Enumerator:

- Valid* identity is verified
- HostMismatch* valid cert provided, but wrong owner
- InvalidCertificate* invalid cert
- NoCertificate* identity unknown

10.69.3 Constructor & Destructor Documentation

10.69.3.1 [QCA::TLS::TLS](#) ([QObject](#) * *parent* = 0, const [QString](#) & *provider* = [QString\(\)](#)) [explicit]

Constructor for Transport Layer Security connection.

This produces a Stream (normal TLS) rather than Datagram (DTLS) object. If you want to do DTLS, see below.

Parameters:

- parent* the parent object for this object
- provider* the name of the provider, if a specific provider is required

10.69.3.2 [QCA::TLS::TLS](#) ([Mode](#) *mode*, [QObject](#) * *parent* = 0, const [QString](#) & *provider* = [QString\(\)](#)) [explicit]

Constructor for Transport Layer Security connection.

Parameters:

- mode* the connection Mode
- parent* the parent object for this object
- provider* the name of the provider, if a specific provider is required

10.69.3.3 QCA::TLS::~~TLS ()

Destructor.

10.69.4 Member Function Documentation

10.69.4.1 void QCA::TLS::reset ()

Reset the connection.

10.69.4.2 QStringList QCA::TLS::supportedCipherSuites (const [Version](#) & *version* = TLS_v1) const

Get the list of cipher suites that are available for use.

A cipher suite is a combination of key exchange, encryption and hashing algorithms that are agreed during the initial handshake between client and server.

Parameters:

version the protocol Version that the cipher suites are required for

Returns:

list of the the names of the cipher suites supported.

10.69.4.3 void QCA::TLS::setCertificate (const [CertificateChain](#) & *cert*, const [PrivateKey](#) & *key*)

The local certificate to use.

This is the certificate that will be provided to the peer. This is almost always required on the server side (because the server has to provide a certificate to the client), and may be used on the client side.

Parameters:

cert a chain of certificates that link the host certificate to a trusted root certificate.

key the private key for the certificate chain

10.69.4.4 void QCA::TLS::setCertificate (const [KeyBundle](#) & *kb*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Allows setting a certificate from a [KeyBundle](#).

10.69.4.5 [CertificateCollection](#) QCA::TLS::trustedCertificates () const

Return the trusted certificates set for this object.

10.69.4.6 void QCA::TLS::setTrustedCertificates (const [CertificateCollection](#) & *trusted*)

Set up the set of trusted certificates that will be used to verify that the certificate provided is valid.

Typically, this will be the collection of root certificates from the system, which you can get using [QCA::systemStore\(\)](#), however you may choose to pass whatever certificates match your assurance needs.

Parameters:

trusted a bundle of trusted certificates.

Examples:

[ssltest.cpp](#).

10.69.4.7 void QCA::TLS::setConstraints (SecurityLevel s)

The security level required for this link.

Parameters:

s the level required for this link.

10.69.4.8 void QCA::TLS::setConstraints (int minSSF, int maxSSF)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

minSSF the minimum Security Strength Factor required for this link.

maxSSF the maximum Security Strength Factor required for this link.

10.69.4.9 void QCA::TLS::setConstraints (const QStringList & cipherSuiteList)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

cipherSuiteList a list of the names of cipher suites that can be used for this link.

Note:

the names are the same as the names in the applicable IETF RFCs (or Internet Drafts if there is no applicable RFC).

10.69.4.10 QList<CertificateInfoOrdered> QCA::TLS::issuerList () const

Retrieve the list of allowed issuers by the server, if the server has provided them.

Only DN types will be present.

```
Certificate someCert = ...
PrivateKey someKey = ...

// see if the server will take our cert
CertificateInfoOrdered issuerInfo = someCert.issuerInfoOrdered().dnOnly();
foreach(const CertificateInfoOrdered &info, tls->issuerList())
{
    if(info == issuerInfo)
    {
        // server will accept someCert, let's present it
    }
}
```

```
        tls->setCertificate(someCert, someKey);  
        break;  
    }  
}
```

Examples:

[ssltest.cpp](#).

10.69.4.11 void QCA::TLS::setIssuerList (const QList< [CertificateInfoOrdered](#) > & issuers)

Sets the issuer list to present to the client.

For use with servers only. Only DN types are allowed.

10.69.4.12 void QCA::TLS::setSession (const [TLSSession](#) & session)

Resume a [TLS](#) session using the given session object.

10.69.4.13 bool QCA::TLS::canCompress () const

Test if the link can use compression.

Returns:

true if the link can use compression

10.69.4.14 bool QCA::TLS::canSetHostName () const

Test if the link can specify a hostname (Server Name Indication).

Returns:

true if the link can specify a hostname

10.69.4.15 bool QCA::TLS::compressionEnabled () const

Returns true if compression is enabled.

This only indicates whether or not the object is configured to use compression, not whether or not the link is actually compressed. Use [isCompressed\(\)](#) for that.

10.69.4.16 void QCA::TLS::setCompressionEnabled (bool b)

Set the link to use compression.

Parameters:

b true if the link should use compression, or false to disable compression

10.69.4.17 `QString QCA::TLS::hostName () const`

Returns the host name specified or an empty string if no host name is specified.

10.69.4.18 `void QCA::TLS::startClient (const QString & host = QString ())`

Start the TLS/SSL connection as a client.

Typically, you'll want to perform RFC 2818 validation on the server's certificate, based on the hostname you're intending to connect to. Pass a value for *host* in order to have the validation for you. If you want to bypass this behavior and do the validation yourself, pass an empty string for *host*.

If the host is an internationalized domain name, then it must be provided in unicode format, not in IDNA ACE/punycode format.

Parameters:

host the hostname that you want to connect to

Note:

The hostname will be used for Server Name Indication extension (see [RFC 3546](#) Section 3.1) if supported by the backend provider.

Examples:

[ssltest.cpp](#).

10.69.4.19 `void QCA::TLS::startServer ()`

Start the TLS/SSL connection as a server.

10.69.4.20 `void QCA::TLS::continueAfterStep ()`

Resumes [TLS](#) processing.

Call this function after `hostNameReceived()`, `certificateRequested()`, `peerCertificateAvailable()` or `handshaken()` is emitted. By requiring this function to be called in order to proceed, applications are given a chance to perform user interaction between steps in the [TLS](#) process.

Examples:

[ssltest.cpp](#).

10.69.4.21 `bool QCA::TLS::isHandshaken () const`

test if the handshake is complete

Returns:

true if the handshake is complete

See also:

[handshaken](#)

10.69.4.22 bool QCA::TLS::isCompressed () const

test if the link is compressed

Returns:

true if the link is compressed

10.69.4.23 Version QCA::TLS::version () const

The protocol version that is in use for this connection.

10.69.4.24 QString QCA::TLS::cipherSuite () const

The cipher suite that has been negotiated for this connection.

The name returned here is the name used in the applicable RFC (or Internet Draft, where there is no RFC).

Examples:

[ssltest.cpp](#).

10.69.4.25 int QCA::TLS::cipherBits () const

The number of effective bits of security being used for this connection.

This can differ from the actual number of bits in the cipher for certain older "export ciphers" that are deliberately crippled. If you want that information, use [cipherMaxBits\(\)](#).

Examples:

[ssltest.cpp](#).

10.69.4.26 int QCA::TLS::cipherMaxBits () const

The number of bits of security that the cipher could use.

This is normally the same as [cipherBits\(\)](#), but can be greater for older "export ciphers".

Examples:

[ssltest.cpp](#).

10.69.4.27 TLSSession QCA::TLS::session () const

The session object of the [TLS](#) connection, which can be used for resuming.

10.69.4.28 Error QCA::TLS::errorCode () const

This method returns the type of error that has occurred.

You should only need to check this if the [error\(\)](#) signal is emitted.

Examples:

[ssltest.cpp](#).

10.69.4.29 IdentityResult QCA::TLS::peerIdentityResult () const

After the SSL/TLS handshake is complete, this method allows you to determine if the other end of the connection (if the application is a client, this is the server; if the application is a server, this is the client) has a valid identity.

Note that the security of TLS/SSL depends on checking this. It is not enough to check that the certificate is valid - you must check that the certificate is valid for the entity that you are trying to communicate with.

Note:

If this returns [QCA::TLS::InvalidCertificate](#), you may wish to use [peerCertificateValidity\(\)](#) to determine whether to proceed or not.

Examples:

[ssltest.cpp](#).

10.69.4.30 Validity QCA::TLS::peerCertificateValidity () const

After the SSL/TLS handshake is valid, this method allows you to check if the received certificate from the other end is valid.

As noted in [peerIdentityResult\(\)](#), you also need to check that the certificate matches the entity you are trying to communicate with.

Examples:

[ssltest.cpp](#).

10.69.4.31 CertificateChain QCA::TLS::localCertificateChain () const

The [CertificateChain](#) for the local host certificate.

10.69.4.32 PrivateKey QCA::TLS::localPrivateKey () const

The [PrivateKey](#) for the local host certificate.

10.69.4.33 CertificateChain QCA::TLS::peerCertificateChain () const

The [CertificateChain](#) from the peer (other end of the connection to the trusted root certificate).

Examples:

[ssltest.cpp](#).

10.69.4.34 virtual bool QCA::TLS::isClosable () const [virtual]

Returns true if the layer has a meaningful "close".

Reimplemented from [QCA::SecureLayer](#).

10.69.4.35 virtual int QCA::TLS::bytesAvailable () const [virtual]

Returns the number of bytes available to be [read\(\)](#) on the application side.

Implements [QCA::SecureLayer](#).

10.69.4.36 virtual int QCA::TLS::bytesOutgoingAvailable () const [virtual]

Returns the number of bytes available to be [readOutgoing\(\)](#) on the network side.

Implements [QCA::SecureLayer](#).

10.69.4.37 virtual void QCA::TLS::close () [virtual]

Close the link.

Note that this may not be meaningful / possible for all implementations.

See also:

[isClosable\(\)](#) for a test that verifies if the link can be closed.

Reimplemented from [QCA::SecureLayer](#).

10.69.4.38 virtual void QCA::TLS::write (const QByteArray & a) [virtual]

This method writes unencrypted (plain) data to the [SecureLayer](#) implementation.

You normally call this function on the application side.

Implements [QCA::SecureLayer](#).

Examples:

[ssltest.cpp](#).

10.69.4.39 virtual QByteArray QCA::TLS::read () [virtual]

This method reads decrypted (plain) data from the [SecureLayer](#) implementation.

You normally call this function on the application side after receiving the [readyRead\(\)](#) signal.

Implements [QCA::SecureLayer](#).

Examples:

[ssltest.cpp](#).

10.69.4.40 virtual void QCA::TLS::writeIncoming (const QByteArray & a) [virtual]

This method accepts encoded (typically encrypted) data for processing.

You normally call this function using data read from the network socket (e.g. using [QTcpSocket::readAll\(\)](#)) after receiving a signal that indicates that the socket has data to read.

Implements [QCA::SecureLayer](#).

Examples:

[ssltest.cpp](#).

10.69.4.41 virtual QByteArray QCA::TLS::readOutgoing (int * *plainBytes* = 0) [virtual]

This method provides encoded (typically encrypted) data.

You normally call this function to get data to write out to the network socket (e.g. using QTcpSocket::write()) after receiving the [readyReadOutgoing\(\)](#) signal.

Implements [QCA::SecureLayer](#).

Examples:

[ssltest.cpp](#).

10.69.4.42 virtual QByteArray QCA::TLS::readUnprocessed () [virtual]

This allows you to read data without having it decrypted first.

This is intended to be used for protocols that close off the connection and return to plain text transfer. You do not normally need to use this function.

Reimplemented from [QCA::SecureLayer](#).

10.69.4.43 virtual int QCA::TLS::convertBytesWritten (qint64 *encryptedBytes*) [virtual]

Convert encrypted bytes written to plain text bytes written.

Implements [QCA::SecureLayer](#).

10.69.4.44 int QCA::TLS::packetsAvailable () const

Determine the number of packets available to be read on the application side.

Note:

this is only used with DTLS.

10.69.4.45 int QCA::TLS::packetsOutgoingAvailable () const

Determine the number of packets available to be read on the network side.

Note:

this is only used with DTLS.

10.69.4.46 int QCA::TLS::packetMTU () const

Return the currently configured maximum packet size.

Note:

this is only used with DTLS

10.69.4.47 void QCA::TLS::setPacketMTU (int *size*) const

Set the maximum packet size to use.

Parameters:

size the number of bytes to set as the MTU.

Note:

this is only used with DTLS.

10.69.4.48 void QCA::TLS::certificateRequested ()

Emitted when the server requests a certificate.

At this time, the client can inspect the [issuerList\(\)](#).

You must call [continueAfterStep\(\)](#) in order for [TLS](#) processing to resume after this signal is emitted.

This signal is only emitted in client mode.

See also:

[continueAfterStep](#)

10.69.4.49 void QCA::TLS::peerCertificateAvailable ()

Emitted when a certificate is received from the peer.

At this time, you may inspect [peerIdentityResult\(\)](#), [peerCertificateValidity\(\)](#), and [peerCertificateChain\(\)](#).

You must call [continueAfterStep\(\)](#) in order for [TLS](#) processing to resume after this signal is emitted.

See also:

[continueAfterStep](#)

10.69.4.50 void QCA::TLS::handshaken ()

Emitted when the protocol handshake is complete.

At this time, all available information about the [TLS](#) session can be inspected.

You must call [continueAfterStep\(\)](#) in order for [TLS](#) processing to resume after this signal is emitted.

See also:

[continueAfterStep](#)
[isHandshaken](#)

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

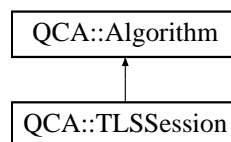
- [qca_securelayer.h](#)

10.70 QCA::TLSSession Class Reference

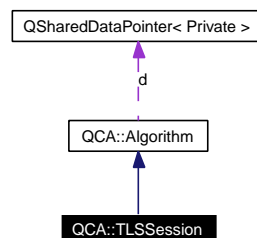
Session token, used for [TLS](#) resuming.

```
#include <qca_securelayer.h>
```

Inheritance diagram for QCA::TLSSession::



Collaboration diagram for QCA::TLSSession:



Public Member Functions

- **TLSSession** (const [TLSSession](#) &from)
- [TLSSession](#) & **operator=** (const [TLSSession](#) &from)
- bool **isNull** () const

10.70.1 Detailed Description

Session token, used for [TLS](#) resuming.

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

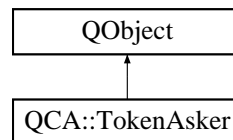
- [qca_securelayer.h](#)

10.71 QCA::TokenAsker Class Reference

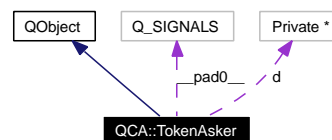
User token handler.

```
#include <qca_core.h>
```

Inheritance diagram for QCA::TokenAsker::



Collaboration diagram for QCA::TokenAsker:



Public Member Functions

- [TokenAsker](#) ([QObject](#) *parent=0)
- void [ask](#) (const [KeyStoreInfo](#) &keyStoreInfo, const [KeyStoreEntry](#) &keyStoreEntry, void *ptr)
- void [cancel](#) ()
- void [waitForResponse](#) ()
- bool [accepted](#) () const

Public Attributes

- [Q_SIGNALS](#) [__pad0__](#): void responseReady()

Friends

- class [Private](#)

10.71.1 Detailed Description

User token handler.

This class is used to request the user to insert a token.

Examples:

[eventhandlerdemo.cpp](#).

10.71.2 Constructor & Destructor Documentation

10.71.2.1 QCA::TokenAsker::TokenAsker (QObject * *parent* = 0)

Construct a new asker.

Parameters:

parent the parent object for this **QObject**

10.71.3 Member Function Documentation

10.71.3.1 void QCA::TokenAsker::ask (const **KeyStoreInfo** & *keyStoreInfo*, const **KeyStoreEntry** & *keyStoreEntry*, void * *ptr*)

queue a token request associated with a key store

Parameters:

keyStoreInfo info of the key store that the information is required for

keyStoreEntry the item in the key store that the information is required for (if applicable)

ptr opaque data

Examples:

[eventhandlerdemo.cpp](#).

10.71.3.2 void QCA::TokenAsker::cancel ()

Cancel the pending password / passphrase request.

10.71.3.3 void QCA::TokenAsker::waitForResponse ()

Block until the token request is completed.

You can use the responseReady signal instead of blocking, if appropriate.

Examples:

[eventhandlerdemo.cpp](#).

10.71.3.4 bool QCA::TokenAsker::accepted () const

Test if the token request was accepted or not.

Returns:

true if the token request was accepted

Examples:

[eventhandlerdemo.cpp](#).

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

- [qca_core.h](#)

Chapter 11

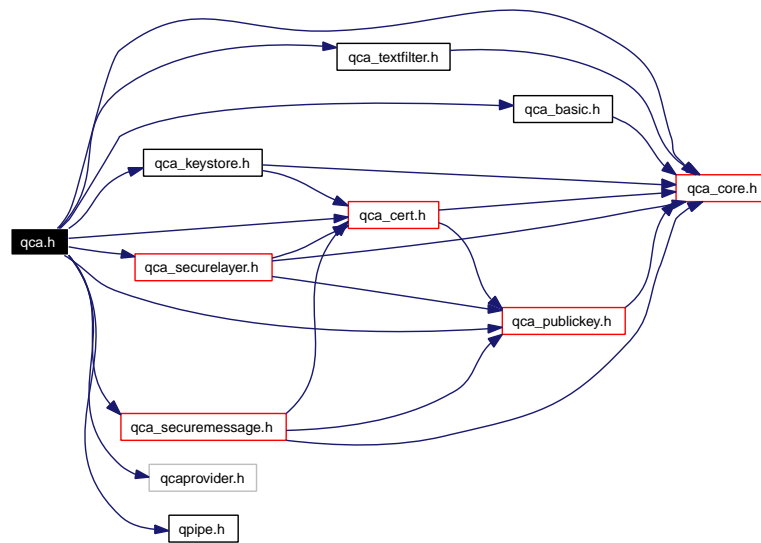
Qt Cryptographic Architecture File Documentation

11.1 qca.h File Reference

Summary header file for QCA.

```
#include "qca_core.h"
#include "qca_textfilter.h"
#include "qca_basic.h"
#include "qca_publickey.h"
#include "qca_cert.h"
#include "qca_keystore.h"
#include "qca_securelayer.h"
#include "qca_securemessage.h"
#include "qcaprovider.h"
#include "qpipe.h"
```

Include dependency graph for qca.h:



11.1.1 Detailed Description

Summary header file for QCA.

Note:

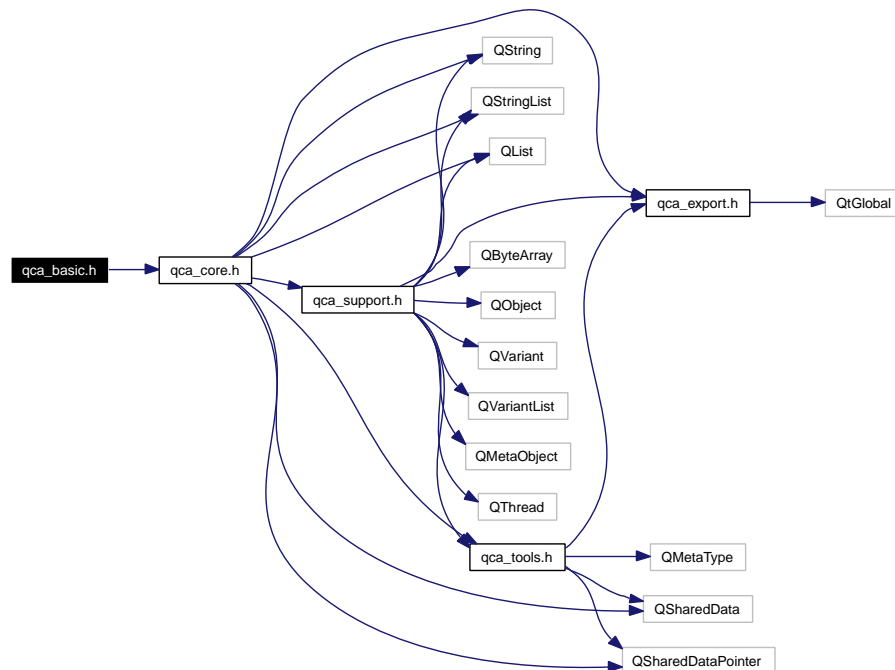
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.2 qca_basic.h File Reference

Header file for classes for cryptographic primitives (basic operations).

```
#include "qca_core.h"
```

Include dependency graph for qca_basic.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::Random](#)
Source of random numbers.
- class [QCA::Hash](#)
General class for hashing algorithms.
- class [QCA::Cipher](#)
General class for cipher (encryption / decryption) algorithms.

- class [QCA::MessageAuthenticationCode](#)
General class for message authentication code (MAC) algorithms.
- class [QCA::KeyDerivationFunction](#)
General superclass for key derivation algorithms.
- class [QCA::PBKDF1](#)
Password based key derivation function version 1.
- class [QCA::PBKDF2](#)
Password based key derivation function version 2.

11.2.1 Detailed Description

Header file for classes for cryptographic primitives (basic operations).

Note:

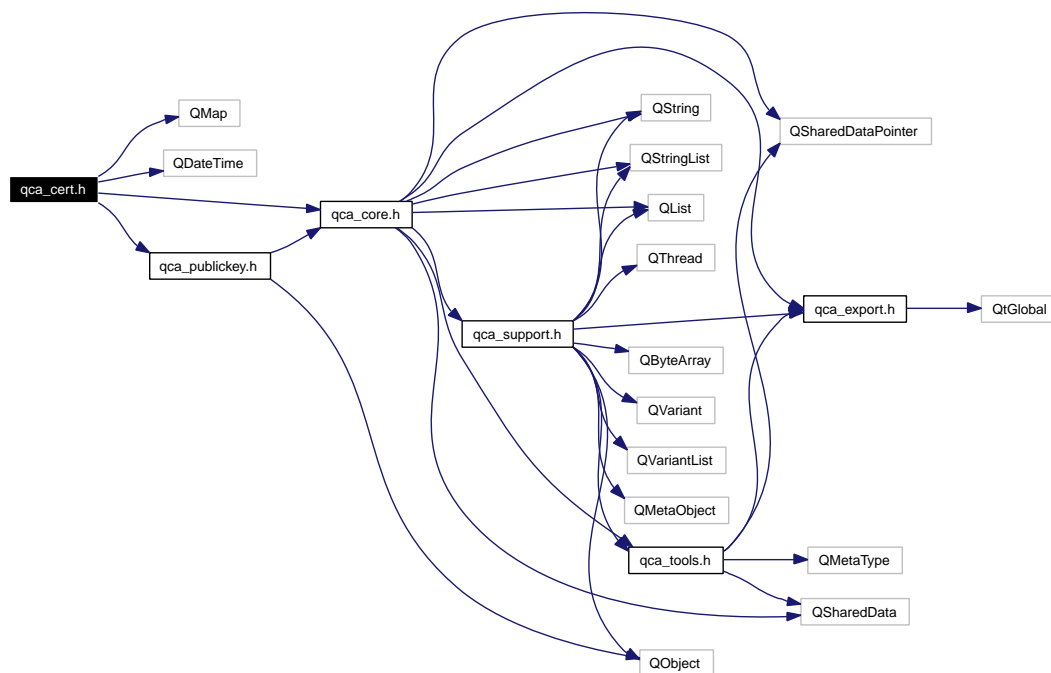
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.3 qca_cert.h File Reference

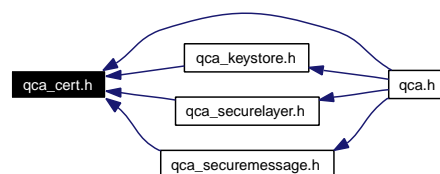
Header file for PGP key and X.509 certificate related classes.

```
#include <QMap>
#include <QDateTime>
#include "qca_core.h"
#include "qca_publickey.h"
```

Include dependency graph for qca_cert.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::CertificateInfoType](#)

Certificate information type.

- class [QCA::CertificateInfoPair](#)
One entry in a certificate information list.
- class [QCA::ConstraintType](#)
Certificate constraint.
- class [QCA::CertificateInfoOrdered](#)
Ordered certificate properties type.
- class [QCA::CertificateOptions](#)
Certificate options
- class [QCA::Certificate](#)
Public Key (X.509) certificate.
- class [QCA::CertificateChain](#)
A chain of related Certificates.
- class [QCA::CertificateRequest](#)
Certificate Request
- class [QCA::CRLEntry](#)
Part of a CRL representing a single certificate.
- class [QCA::CRL](#)
Certificate Revocation List
- class [QCA::CertificateCollection](#)
Bundle of Certificates and CRLs.
- class [QCA::CertificateAuthority](#)
A Certificate Authority is used to generate Certificates and Certificate Revocation Lists (CRLs).
- class [QCA::KeyBundle](#)
Certificate chain and private key pair.
- class [QCA::PGPKey](#)
Pretty Good Privacy key.
- class [QCA::KeyLoader](#)
Asynchronous private key loader.

Typedefs

- typedef [QMultiMap](#)< CertificateInfoType, [QString](#) > [QCA::CertificateInfo](#)
- typedef [QList](#)< ConstraintType > [QCA::Constraints](#)

Enumerations

- enum [QCA::CertificateRequestFormat](#) { [QCA::PKCS10](#), [QCA::SPKAC](#) }
- enum [QCA::CertificateInfoTypeKnown](#) {
[QCA::CommonName](#), [QCA::Email](#), [QCA::EmailLegacy](#), [QCA::Organization](#),
[QCA::OrganizationalUnit](#), [QCA::Locality](#), [QCA::IncorporationLocality](#), [QCA::State](#),
[QCA::IncorporationState](#), [QCA::Country](#), [QCA::IncorporationCountry](#), [QCA::URI](#),
[QCA::DNS](#), [QCA::IPAddress](#), [QCA::XMPP](#) }
- enum [QCA::ConstraintTypeKnown](#) {
[QCA::DigitalSignature](#), [QCA::NonRepudiation](#), [QCA::KeyEncipherment](#), [QCA::Data-Encipherment](#),
[QCA::KeyAgreement](#), [QCA::KeyCertificateSign](#), [QCA::CRLSign](#), [QCA::EncipherOnly](#),
[QCA::DecipherOnly](#), [QCA::ServerAuth](#), [QCA::ClientAuth](#), [QCA::CodeSigning](#),
[QCA::EmailProtection](#), [QCA::IPSecEndSystem](#), [QCA::IPSecTunnel](#), [QCA::IPSecUser](#),
[QCA::TimeStamping](#), [QCA::OCSPSigning](#) }
- enum [QCA::UsageMode](#) {
[QCA::UsageAny](#) = 0x00, [QCA::UsageTLSServer](#) = 0x01, [QCA::UsageTLSClient](#) = 0x02,
[QCA::UsageCodeSigning](#) = 0x04,
[QCA::UsageEmailProtection](#) = 0x08, [QCA::UsageTimeStamping](#) = 0x10, [QCA::UsageCRLSigning](#)
= 0x20 }
- enum [QCA::Validity](#) {
[QCA::ValidityGood](#), [QCA::ErrorRejected](#), [QCA::ErrorUntrusted](#), [QCA::ErrorSignatureFailed](#),
[QCA::ErrorInvalidCA](#), [QCA::ErrorInvalidPurpose](#), [QCA::ErrorSelfSigned](#), [QCA::ErrorRevoked](#),
[QCA::ErrorPathLengthExceeded](#), [QCA::ErrorExpired](#), [QCA::ErrorExpiredCA](#), [QCA::Error-ValidityUnknown](#) = 64 }
- enum [QCA::ValidateFlags](#) { [ValidateAll](#) = 0x00, [ValidateRevoked](#) = 0x01, [ValidateExpired](#) = 0x02, [ValidatePolicy](#) = 0x04 }

Functions

- [QCA_EXPORT QString](#) [QCA::orderedToDNString](#) (const CertificateInfoOrdered &in)
- [QCA_EXPORT CertificateInfoOrdered](#) [QCA::orderedDNOnly](#) (const CertificateInfoOrdered &in)
- [QCA_EXPORT QList](#) [QCA::makeFriendlyNames](#) (const [QList](#)< Certificate > &list)

11.3.1 Detailed Description

Header file for PGP key and X.509 certificate related classes.

Note:

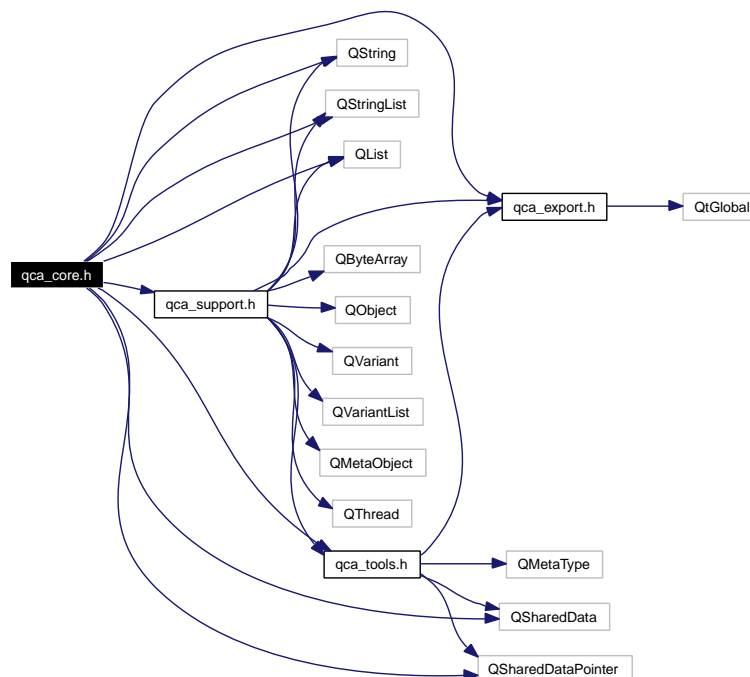
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.4 qca_core.h File Reference

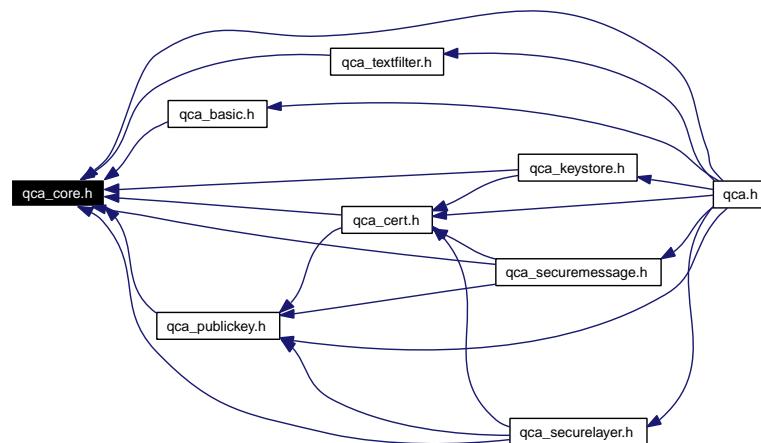
Header file for core QCA infrastructure.

```
#include <QString>
#include <QStringList>
#include <QList>
#include <QSharedData>
#include <QSharedDataPointer>
#include "qca_export.h"
#include "qca_support.h"
#include "qca_tools.h"
```

Include dependency graph for qca_core.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::Initializer](#)
Convenience method for initialising and cleaning up QCA.
- class [QCA::KeyLength](#)
Simple container for acceptable key lengths.
- class [QCA::Provider](#)
- class [QCA::Provider](#)
Algorithm provider.
- class [QCA::BasicContext](#)
Base class to use for primitive provider contexts.
- class [QCA::BufferedComputation](#)
General superclass for buffered computation algorithms.
- class [QCA::Filter](#)
General superclass for filtering transformation algorithms.
- class [QCA::Algorithm](#)
General superclass for an algorithm.
- class [QCA::SymmetricKey](#)
Container for keys for symmetric encryption algorithms.
- class [QCA::InitializationVector](#)

Container for initialisation vectors and nonces.

- class [QCA::Event](#)
An asynchronous event.
- class [QCA::EventHandler](#)
Interface class for password / passphrase / PIN and token handlers.
- class [QCA::PasswordAsker](#)
User password / passphrase / PIN handler.
- class [QCA::TokenAsker](#)
User token handler.

Defines

- #define [QCA_VERSION](#) 0x016363
- #define [QCA_logTextMessage](#)(message, severity)
- #define [QCA_logBinaryMessage](#)(blob, severity)

Typedefs

- typedef [QList](#)< Provider * > [QCA::ProviderList](#)

Enumerations

- enum [QCA::MemoryMode](#) { [QCA::Practical](#), [QCA::Locking](#), [QCA::LockingKeepPrivileges](#) }
- enum [QCA::Direction](#) { [QCA::Encode](#), [QCA::Decode](#) }

Functions

- QCA_EXPORT int [qcaVersion](#) ()
- QCA_EXPORT void [QCA::init](#) ()
- QCA_EXPORT void [QCA::init](#) ([MemoryMode](#) m, int prealloc)
- QCA_EXPORT void [QCA::deinit](#) ()
- QCA_EXPORT bool [QCA::haveSecureMemory](#) ()
- QCA_EXPORT bool [QCA::haveSecureRandom](#) ()
- QCA_EXPORT bool [QCA::isSupported](#) (const char *features, const [QString](#) &provider=[QString](#)())
- QCA_EXPORT bool [QCA::isSupported](#) (const [QStringList](#) &features, const [QString](#) &provider=[QString](#)())
- QCA_EXPORT [QStringList](#) [QCA::supportedFeatures](#) ()
- QCA_EXPORT [QStringList](#) [QCA::defaultFeatures](#) ()
- QCA_EXPORT bool [QCA::insertProvider](#) (Provider *p, int priority=0)
- QCA_EXPORT void [QCA::setProviderPriority](#) (const [QString](#) &name, int priority)
- QCA_EXPORT int [QCA::providerPriority](#) (const [QString](#) &name)
- QCA_EXPORT [ProviderList](#) [QCA::providers](#) ()
- QCA_EXPORT Provider * [QCA::findProvider](#) (const [QString](#) &name)

- QCA_EXPORT Provider * [QCA::defaultProvider](#) ()
- QCA_EXPORT void [QCA::scanForPlugins](#) ()
- QCA_EXPORT void [QCA::unloadAllPlugins](#) ()
- QCA_EXPORT **QString** [QCA::pluginDiagnosticText](#) ()
- QCA_EXPORT void [QCA::clearPluginDiagnosticText](#) ()
- QCA_EXPORT void [QCA::appendPluginDiagnosticText](#) (const **QString** &text)
- QCA_EXPORT void [QCA::setProperty](#) (const **QString** &name, const **QVariant** &value)
- QCA_EXPORT **QVariant** [QCA::getProperty](#) (const **QString** &name)
- QCA_EXPORT void [QCA::setProviderConfig](#) (const **QString** &name, const QVariantMap &config)
- QCA_EXPORT QVariantMap [QCA::getProviderConfig](#) (const **QString** &name)
- QCA_EXPORT void [QCA::saveProviderConfig](#) (const **QString** &name)
- QCA_EXPORT **QString** [QCA::globalRandomProvider](#) ()
- QCA_EXPORT void [QCA::setGlobalRandomProvider](#) (const **QString** &provider)
- QCA_EXPORT Logger * [QCA::logger](#) ()
- QCA_EXPORT bool [QCA::haveSystemStore](#) ()
- QCA_EXPORT CertificateCollection [QCA::systemStore](#) ()
- QCA_EXPORT **QString** [QCA::appName](#) ()
- QCA_EXPORT void [QCA::setAppName](#) (const **QString** &name)
- QCA_EXPORT **QString** [QCA::arrayToHex](#) (const **QByteArray** &array)
- QCA_EXPORT **QByteArray** [QCA::hexToArray](#) (const **QString** &hexString)

11.4.1 Detailed Description

Header file for core QCA infrastructure.

Note:

You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.4.2 Define Documentation

11.4.2.1 `#define QCA_VERSION 0x016363`

The current version of QCA.

This provides you a compile time check of the QCA version.

See also:

[qcaVersion](#) for a runtime check.

Examples:

[aes-cmac.cpp](#).

11.4.2.2 `#define QCA_logTextMessage(message, severity)`

Value:

```
do { \
    register QCA::Logger::Severity s = severity; \
    register QCA::Logger *l = QCA::logger (); \
    if (s <= l->level ()) { \
        l->logTextMessage (message, s); \
    } \
} while (false)
```

Log a text message.

This is an efficient function to avoid overhead of argument executions when log level blocks the message.

Parameters:

message the text to log

severity the type of information to log

Note:

This is a macro, so arguments may or may not be evaluated.

11.4.2.3 #define QCA_logBinaryMessage(blob, severity)

Value:

```
do { \
    register QCA::Logger::Severity s = severity; \
    register QCA::Logger *l = QCA::logger (); \
    if (s <= l->level ()) { \
        l->logBinaryMessage (blob, s); \
    } \
} while (false)
```

Log a binary message.

This is an efficient function to avoid overhead of argument executions when log level blocks the message.

Parameters:

blob the blob to log

severity the type of information to log

Note:

This is a macro, so arguments may or may not be evaluated.

11.4.3 Function Documentation

11.4.3.1 QCA_EXPORT int qcaVersion ()

The current version of QCA.

This is equivalent to QCA_VERSION, except it provides a runtime check of the version of QCA that is being used.

Examples:

[aes-cmac.cpp](#).

11.5 qca_export.h File Reference

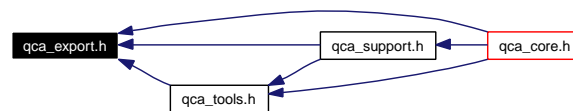
Preprocessor magic to allow export of library symbols.

```
#include <QtGlobal>
```

Include dependency graph for qca_export.h:



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



11.5.1 Detailed Description

Preprocessor magic to allow export of library symbols.

This is strictly internal.

Note:

You should not include this header directly from an application. You should just use `#include <QtCrypto>` instead.

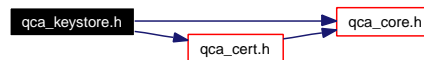
11.6 qca_keystore.h File Reference

Header file for classes that provide and manage keys.

```
#include "qca_core.h"
```

```
#include "qca_cert.h"
```

Include dependency graph for qca_keystore.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::KeyStoreEntry](#)
Single entry in a [KeyStore](#).
- class [QCA::KeyStoreEntryWatcher](#)
Class to monitor the availability of a [KeyStoreEntry](#).
- class [QCA::KeyStore](#)
General purpose key storage object.
- class [QCA::KeyStoreInfo](#)
Key store information, outside of a [KeyStore](#) object.
- class [QCA::KeyStoreManager](#)
Access keystores, and monitor keystores for changes.

11.6.1 Detailed Description

Header file for classes that provide and manage keys.

Note:

You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

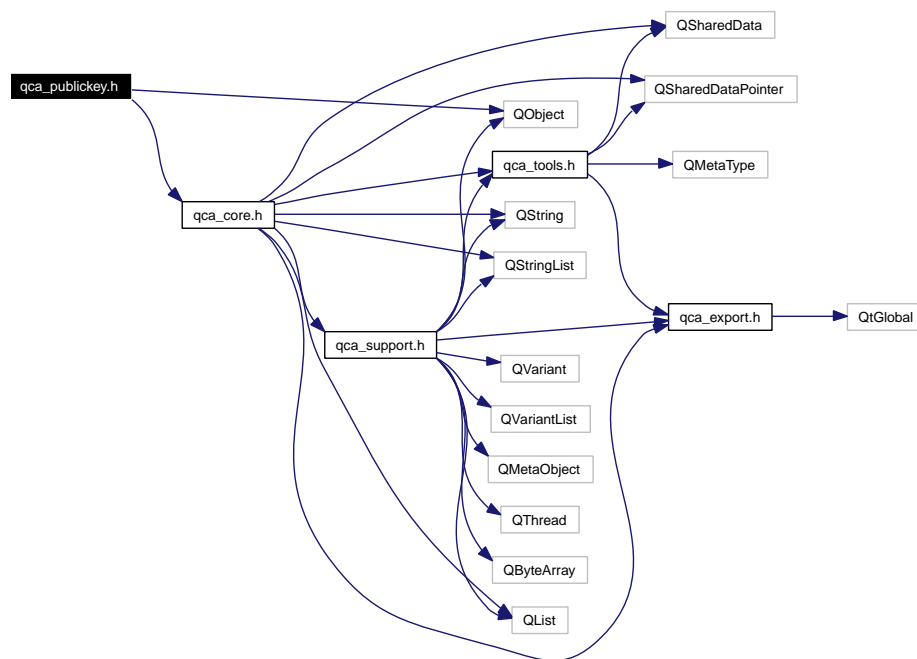
11.7 qca_publickey.h File Reference

Header file for PublicKey and PrivateKey related classes.

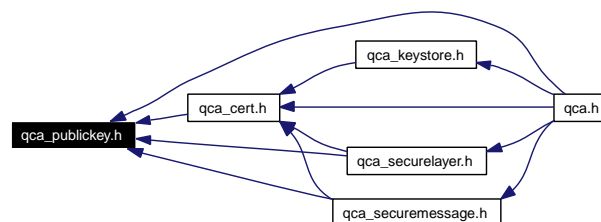
```
#include <QObject>
```

```
#include "qca_core.h"
```

Include dependency graph for qca_publickey.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::DLGroup](#)
A discrete logarithm group.
- class [QCA::PKey](#)
General superclass for public ([PublicKey](#)) and private ([PrivateKey](#)) keys used with asymmetric encryption techniques.
- class [QCA::PublicKey](#)
Generic public key.
- class [QCA::PrivateKey](#)
Generic private key.
- class [QCA::KeyGenerator](#)
Class for generating asymmetric key pairs.
- class [QCA::RSAPublicKey](#)
RSA Public Key.
- class [QCA::RSAPrivateKey](#)
RSA Private Key.
- class [QCA::DSAPublicKey](#)
Digital Signature Algorithm Public Key.
- class [QCA::DSAPrivateKey](#)
Digital Signature Algorithm Private Key.
- class [QCA::DHPublicKey](#)
Diffie-Hellman Public Key.
- class [QCA::DHPrivateKey](#)
Diffie-Hellman Private Key.

Enumerations

- enum [QCA::EncryptionAlgorithm](#) { [QCA::EME_PKCS1v15](#), [QCA::EME_PKCS1_OAEP](#) }
- enum [QCA::SignatureAlgorithm](#) {
[QCA::SignatureUnknown](#), [QCA::EMSA1_SHA1](#), [QCA::EMSA3_SHA1](#), [QCA::EMSA3_MD5](#),
[QCA::EMSA3_MD2](#), [QCA::EMSA3_RIPEMD160](#), [QCA::EMSA3_Raw](#) }
- enum [QCA::SignatureFormat](#) { [QCA::DefaultFormat](#), [QCA::IEEE_1363](#), [QCA::DERSequence](#) }
- enum [QCA::PBEAlgorithm](#) {
[QCA::PBEDefault](#), [QCA::PBES2_DES_SHA1](#), [QCA::PBES2_TripleDES_SHA1](#), [QCA::PBES2_-AES128_SHA1](#),
[QCA::PBES2_AES192_SHA1](#), [QCA::PBES2_AES256_SHA1](#) }

- enum `QCA::ConvertResult` { `QCA::ConvertGood`, `QCA::ErrorDecode`, `QCA::ErrorPassphrase`, `QCA::ErrorFile` }
- enum `QCA::DLGroupSet` {
`QCA::DSA_512`, `QCA::DSA_768`, `QCA::DSA_1024`, `QCA::IETF_768`,
`QCA::IETF_1024`, `QCA::IETF_1536`, `QCA::IETF_2048`, `QCA::IETF_3072`,
`QCA::IETF_4096`, `QCA::IETF_6144`, `QCA::IETF_8192` }

Functions

- `QCA_EXPORT QByteArray QCA::emsa3Encode` (const `QString` &hashName, const `QByteArray` &digest, int size=-1)

11.7.1 Detailed Description

Header file for `PublicKey` and `PrivateKey` related classes.

Note:

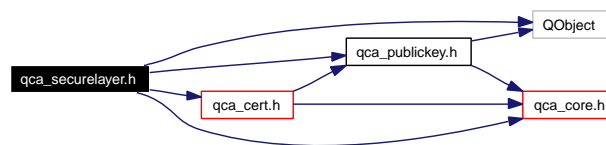
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.8 qca_securelayer.h File Reference

Header file for SecureLayer and its subclasses.

```
#include <QObject>
#include "qca_core.h"
#include "qca_publickey.h"
#include "qca_cert.h"
```

Include dependency graph for qca_securelayer.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::SecureLayer](#)
Abstract interface to a security layer.
- class [QCA::TLSSession](#)
Session token, used for [TLS](#) resuming.
- class [QCA::TLS](#)
Transport Layer Security / Secure Socket Layer.
- class [QCA::SASL](#)
Simple Authentication and Security Layer protocol implementation.
- class [QCA::SASL::Params](#)
Parameter flags for the [SASL](#) authentication.

Enumerations

- enum [QCA::SecurityLevel](#) {

QCA::SL_None, QCA::SL_Integrity, QCA::SL_Export, QCA::SL_Baseline,
QCA::SL_High, QCA::SL_Highest }

11.8.1 Detailed Description

Header file for SecureLayer and its subclasses.

Note:

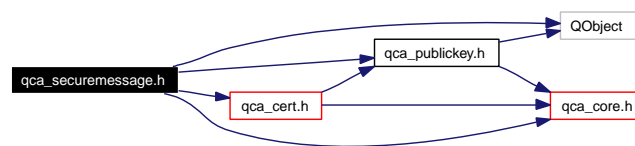
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.9 qca_securemessage.h File Reference

Header file for secure message (PGP, CMS) classes.

```
#include <QObject>
#include "qca_core.h"
#include "qca_publickey.h"
#include "qca_cert.h"
```

Include dependency graph for qca_securemessage.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::SecureMessageKey](#)
Key for [SecureMessage](#) system.
- class [QCA::SecureMessageSignature](#)
[SecureMessage](#) signature.
- class [QCA::SecureMessage](#)
Class representing a secure message.
- class [QCA::SecureMessageSystem](#)
Abstract superclass for secure messaging systems.
- class [QCA::OpenPGP](#)
Pretty Good Privacy messaging system.
- class [QCA::CMS](#)
Cryptographic Message Syntax messaging system.

Typedefs

- typedef **QList**< SecureMessageKey > [QCA::SecureMessageKeyList](#)
- typedef **QList**< SecureMessageSignature > [QCA::SecureMessageSignatureList](#)

11.9.1 Detailed Description

Header file for secure message (PGP, CMS) classes.

Note:

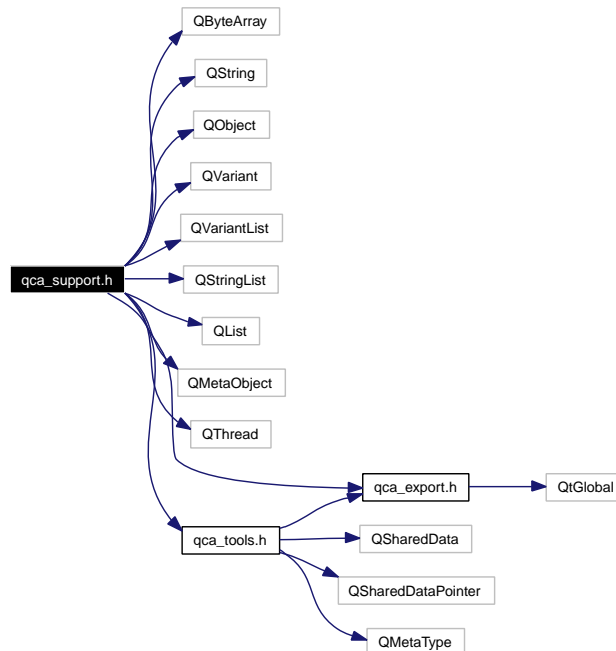
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.10 qca_support.h File Reference

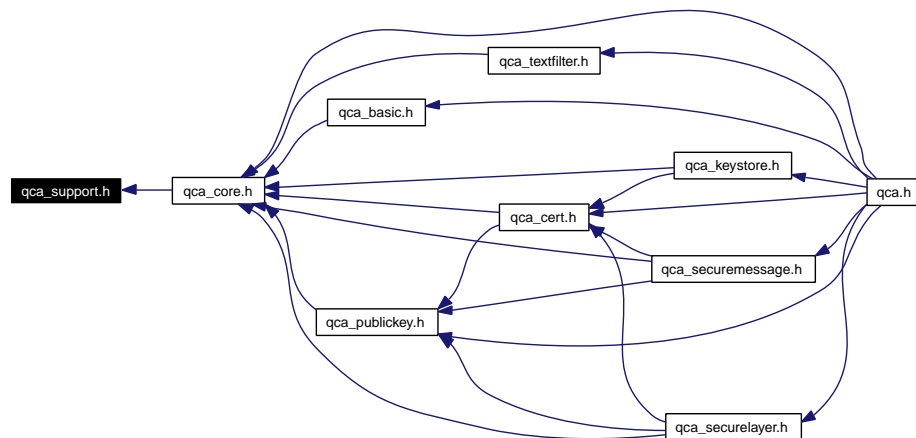
Header file for "support" classes used in QCA.

```
#include <QByteArray>
#include <QString>
#include <QObject>
#include <QVariant>
#include <QVariantList>
#include <QStringList>
#include <QList>
#include <QMetaObject>
#include <QThread>
#include "qca_export.h"
#include "qca_tools.h"
```

Include dependency graph for qca_support.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class **QCA::SyncThread**
- class **QCA::Synchronizer**
- class **QCA::DirWatch**
- class [QCA::FileWatch](#)
Support class to monitor a file for activity.
- class **QCA::Console**
- class **QCA::ConsoleReference**
- class **QCA::ConsolePrompt**
- class [QCA::Logger](#)
A simple logging system.
- class [QCA::AbstractLogDevice](#)
An abstract log device.

Functions

- **QCA_EXPORT QByteArray** [QCA::methodReturnType](#) (const **QMetaObject** *obj, const **QByteArray** &method, const **QList**< **QByteArray** > argTypes)
- **QCA_EXPORT bool** [QCA::invokeMethodWithVariants](#) (**QObject** *obj, const **QByteArray** &method, const **QVariantList** &args, **QVariant** *ret, Qt::ConnectionType type=Qt::AutoConnection)

11.10.1 Detailed Description

Header file for "support" classes used in QCA.

The classes in this header do not have any cryptographic content - they are used in QCA, and are included for convenience.

Note:

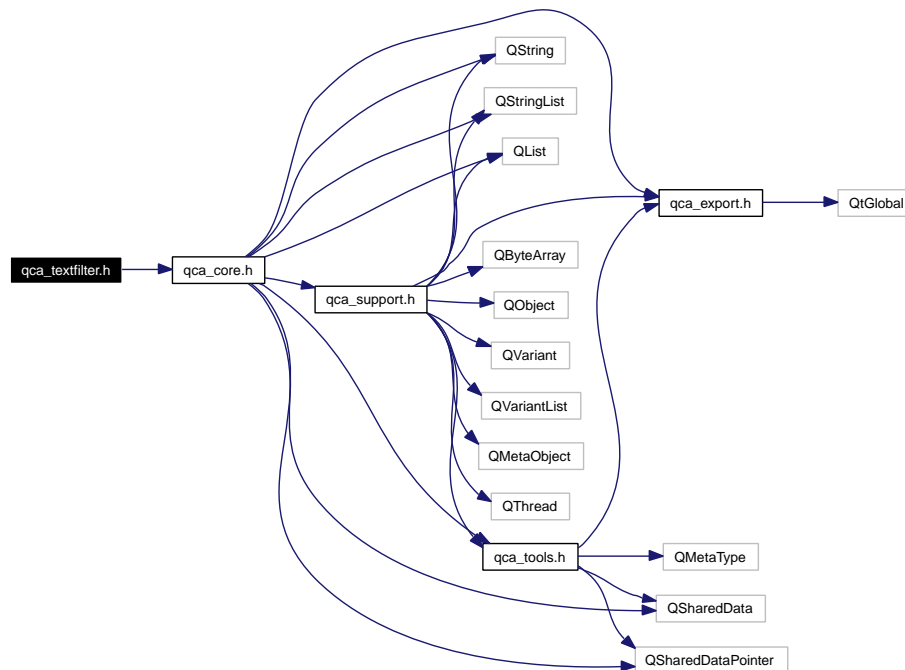
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.11 qca_textfilter.h File Reference

Header file for text encoding/decoding classes.

```
#include "qca_core.h"
```

Include dependency graph for qca_textfilter.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::TextFilter](#)
Superclass for text based filtering algorithms.
- class [QCA::Hex](#)
Hexadecimal encoding / decoding.
- class [QCA::Base64](#)
Base64 encoding / decoding

11.11.1 Detailed Description

Header file for text encoding/decoding classes.

Note:

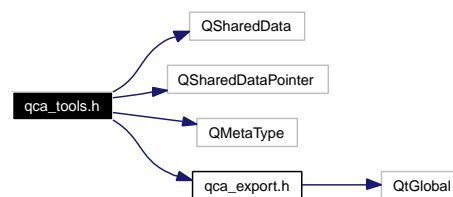
You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.12 qca_tools.h File Reference

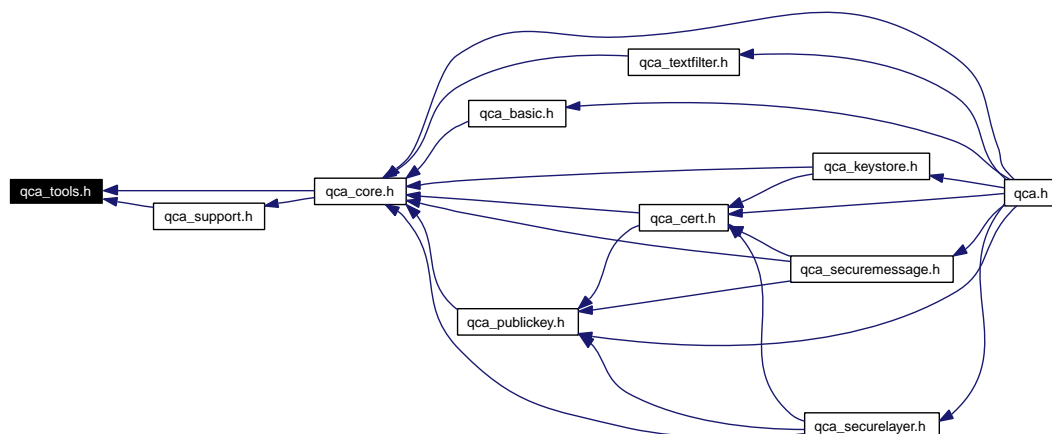
Header file for "tool" classes used in QCA.

```
#include <QSharedData>
#include <QSharedDataPointer>
#include <QMetaType>
#include "qca_export.h"
```

Include dependency graph for qca_tools.h:



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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::MemoryRegion](#)
Array of bytes that may be optionally secured.
- class [QCA::SecureArray](#)
Secure array of bytes.

- class [QCA::BigInteger](#)
Arbitrary precision integer.

Functions

- QCA_EXPORT void * [qca_secure_alloc](#) (int bytes)
- QCA_EXPORT void [qca_secure_free](#) (void *p)
- QCA_EXPORT void * [qca_secure_realloc](#) (void *p, int bytes)
- QCA_EXPORT const SecureArray [QCA::operator+](#) (const SecureArray &a, const SecureArray &b)

11.12.1 Detailed Description

Header file for "tool" classes used in QCA.

These classes differ from those in [qca_support.h](#), in that they have some cryptographic relationship, and require secure memory.

Note:

You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

11.12.2 Function Documentation

11.12.2.1 QCA_EXPORT void* qca_secure_alloc (int bytes)

Allocate a block of memory from the secure memory pool.

This is intended to be used when working with C libraries.

Parameters:

bytes the number of bytes to allocate

11.12.2.2 QCA_EXPORT void qca_secure_free (void *p)

Free (de-allocate) a block of memory that has been previously allocated from the secure memory pool.

This is intended to be used when working with C libraries.

Parameters:

p pointer to the block of memory to be free'd

11.12.2.3 QCA_EXPORT void* qca_secure_realloc (void *p, int bytes)

Resize (re-allocate) a block of memory that has been previously allocated from the secure memory pool.

Parameters:

p pointer to the block of memory to be resized.

bytes the new size that is required.

11.13 qpipe.h File Reference

Header file for the QPipe FIFO class.

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



Namespaces

- namespace [QCA](#)

Classes

- class [QCA::QPipeDevice](#)
- class [QCA::QPipeEnd](#)
- class [QCA::QPipe](#)

A FIFO buffer (named pipe) abstraction.

11.13.1 Detailed Description

Header file for the QPipe FIFO class.

Note:

You should not use this header directly from an application. You should just use `#include <Qt-Crypto>` instead.

Chapter 12

Qt Cryptographic Architecture Example Documentation

12.1 aes-cmac.cpp

This example shows how to implement a client side "provider".

There are three important parts to this:

- the class derived from [QCA::Provider](#) (in this example called "ClientSideProvider"), that generates the context class
- one or more context classes (in this example only one, implementing AES-CMAC, called "AESC-MACContext")
- a call to [QCA::insertProvider](#), to add the [QCA::Provider](#) subclass into [QCA](#)

```
1 /*
2  Copyright (C) 2006 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26 #include <QDebug>
27
```

```
28 class AESCMACContext : public QCA::MACContext
29 {
30 public:
31     AESCMACContext(QCA::Provider *p) : QCA::MACContext(p, "cmac(aes)")
32     {
33     }
34
35     ~AESCMACContext()
36     {
37     }
38
39
40     // Helper to left shift an arbitrary length array
41     // This is heavily based on the example in the I-D.
42     QCA::SecureArray leftShift(const QCA::SecureArray &array)
43     {
44         // We create an output of the same size as the input
45         QCA::SecureArray out(array.size());
46         // We handle one byte at a time - this is the high bit
47         // from the previous byte.
48         int overflow = 0;
49
50         // work through each byte.
51         for (int i = array.size() - 1; i >= 0; --i) {
52             // do the left shift on this byte.
53             out[i] = array[i] << 1;
54             // make the low bit on this byte be the high bit
55             // from the previous byte.
56             out[i] |= overflow;
57             // save the high bit for next time
58             overflow = (array[i] & 0x80) ? 1 : 0;
59         }
60         return out;
61     }
62
63
64     // Helper to XOR two arrays - must be same length
65     QCA::SecureArray xorArray(const QCA::SecureArray &array1,
66                               const QCA::SecureArray &array2)
67     {
68         if (array1.size() != array2.size())
69             // empty array
70             return QCA::SecureArray();
71
72         QCA::SecureArray result(array1.size());
73
74         for (int i = 0; i < array1.size(); ++i)
75             result[i] = array1[i] ^ array2[i];
76
77         return result;
78     }
79
80
81     void setup(const QCA::SymmetricKey &key)
82     {
83         // We might not have a real key, since this can get called
84         // from the constructor.
85         if (key.size() == 0)
86             return;
87
88         m_key = key;
89         // Generate the subkeys
90         QCA::SecureArray const_Zero(16);
91         QCA::SecureArray const_Rb(16);
92         const_Rb[15] = (char)0x87;
93
94         m_X = const_Zero;
```

```

95         m_residual = QCA::SecureArray();
96
97         // Figure 2.2, step 1.
98         QCA::Cipher aesObj(QString("aes128"),
99                             QCA::Cipher::ECB, QCA::Cipher::DefaultPadding,
100                             QCA::Encode, key);
101         QCA::SecureArray L = aesObj.process(const_Zero);
102
103         // Figure 2.2, step 2
104         if (0 == (L[0] & 0x80))
105             m_k1 = leftShift(L);
106         else
107             m_k1 = xorArray(leftShift(L), const_Rb);
108
109         // Figure 2.2, step 3
110         if (0 == (m_k1[0] & 0x80))
111             m_k2 = leftShift(m_k1);
112         else
113             m_k2 = xorArray(leftShift(m_k1), const_Rb);
114     }
115
116     QCA::Provider::Context *clone() const
117     {
118         return new AESCMACContext(*this);
119     }
120
121     void clear()
122     {
123         setup(m_key);
124     }
125
126     QCA::KeyLength keyLength() const
127     {
128         return QCA::KeyLength(16, 16, 1);
129     }
130
131     // This is a bit different to the way the I-D does it,
132     // to allow for multiple update() calls.
133     void update(const QCA::MemoryRegion &a)
134     {
135         QCA::SecureArray bytesToProcess = m_residual + a;
136         int blockNum;
137         // note that we don't want to do the last full block here, because
138         // it needs special treatment in final().
139         for (blockNum = 0; blockNum < ((bytesToProcess.size()-1)/16); ++blockNum) {
140             // copy a block of data
141             QCA::SecureArray thisBlock(16);
142             for (int yalv = 0; yalv < 16; ++yalv)
143                 thisBlock[yalv] = bytesToProcess[blockNum*16 + yalv];
144
145             m_Y = xorArray(m_X, thisBlock);
146
147             QCA::Cipher aesObj(QString("aes128"),
148                                 QCA::Cipher::ECB, QCA::Cipher::DefaultPadding,
149                                 QCA::Encode, m_key);
150             m_X = aesObj.process(m_Y);
151         }
152         // This can be between 1 and 16
153         int numBytesLeft = bytesToProcess.size() - 16*blockNum;
154         // we copy the left over part
155         m_residual.resize(numBytesLeft);
156         for(int yalv = 0; yalv < numBytesLeft; ++yalv)
157             m_residual[yalv] = bytesToProcess[blockNum*16 + yalv];
158     }
159
160     void final( QCA::MemoryRegion *out)
161     {

```

```

162     QCA::SecureArray lastBlock;
163     int numBytesLeft = m_residual.size();
164
165     if ( numBytesLeft != 16 ) {
166         // no full block, so we have to pad.
167         m_residual.resize(16);
168         m_residual[numBytesLeft] = (char)0x80;
169         lastBlock = xorArray(m_residual, m_k2);
170     } else {
171         // this is a full block - no padding
172         lastBlock = xorArray(m_residual, m_k1);
173     }
174     m_Y = xorArray(m_X, lastBlock);
175     QCA::Cipher aesObj(QString("aes128"),
176                         QCA::Cipher::ECB, QCA::Cipher::DefaultPadding,
177                         QCA::Encode, m_key);
178     *out = aesObj.process(m_Y);
179
180 }
181
182 protected:
183     // first subkey
184     QCA::SecureArray m_k1;
185     // second subkey
186     QCA::SecureArray m_k2;
187     // main key
188     QCA::SecureArray m_key;
189
190     // state
191     QCA::SecureArray m_X;
192     QCA::SecureArray m_Y;
193
194     // partial block that we can't do yet
195     QCA::SecureArray m_residual;
196 };
197
198 class ClientSideProvider : public QCA::Provider
199 {
200 public:
201     int qcaVersion() const
202     {
203         return QCA_VERSION;
204     }
205
206     QString name() const
207     {
208         return "exampleClientSideProvider";
209     }
210
211     QStringList features() const
212     {
213         QStringList list;
214         list += "cmac(aes)";
215         // you can add more features in here, if you have some.
216         return list;
217     }
218
219     Provider::Context *createContext(const QString &type)
220     {
221         if(type == "cmac(aes)")
222             return new AESCMACContext(this);
223         // else if (type == some other feature)
224         // return some other context.
225         else
226             return 0;
227     }
228 };

```

```

229
230
231 // AES CMAC is a Message Authentication Code based on a block cipher
232 // instead of the more normal keyed hash.
233 // See RFC 4493 "The AES-CMAC Algorithm"
234 class AES_CMAC: public QCA::MessageAuthenticationCode
235 {
236 public:
237     AES_CMAC(const QCA::SymmetricKey &key = QCA::SymmetricKey(),
238             const QString &provider = QString()):
239         QCA::MessageAuthenticationCode( "cmac(aes)", key, provider)
240     {}
241 };
242
243
244 int main(int argc, char **argv)
245 {
246     QCoreApplication app(argc, argv);
247
248     qDebug() << "This example shows AES CMAC";
249
250     // the Initializer object sets things up, and
251     // also does cleanup when it goes out of scope
252     QCA::Initializer init;
253
254     qDebug() << "Completed initialisation";
255
256     if( ! QCA::isSupported("aes128-ecb") ) {
257         qDebug() << "AES not supported!";
258     }
259
260     if ( QCA::insertProvider(new ClientSideProvider, 0) )
261         qDebug() << "Inserted our provider";
262     else
263         qDebug() << "our provider could not be added";
264
265     // We should check AES CMAC is supported before using it.
266     if( ! QCA::isSupported("cmac(aes)") ) {
267         qDebug() << "AES CMAC not supported!";
268     } else {
269         // create the required object
270         AES_CMAC cmacObject;
271
272         // create the key
273         QCA::SymmetricKey key(QCA::hexToArray("2b7e151628aed2a6abf7158809cf4f3c"));
274
275         // set the MAC to use the key
276         cmacObject.setup(key);
277
278         QCA::SecureArray message = QCA::hexToArray("6bc1bee22e409f96e93d7e117393172a"
279             "ae2d8a571e03ac9c9eb76fac45af8e51"
280             "30c81c46a35ce411e5fbc1191a0a52ef"
281             "f69f2445df4f9b17ad2b417be66c3710");
282
283         QCA::SecureArray message1(message);
284         message1.resize(0);
285         qDebug();
286         qDebug() << "Message1: " << QCA::arrayToHex(message1.toByteArray());
287         qDebug() << "Expecting:  bb1d6929e95937287fa37d129b756746";
288         qDebug() << "AES-CMAC: " << QCA::arrayToHex(cmacObject.process(message1).toByteArray());
289
290         cmacObject.clear();
291         QCA::SecureArray message2(message);
292         message2.resize(16);
293         qDebug();
294         qDebug() << "Message2: " << QCA::arrayToHex(message2.toByteArray());
295         qDebug() << "Expecting:  070a16b46b4d4144f79bdd9dd04a287c";
296         qDebug() << "AES-CMAC: " << QCA::arrayToHex(cmacObject.process(message2).toByteArray());

```

```
296
297     cmacObject.clear();
298     QCA::SecureArray message3(message);
299     message3.resize(40);
300     qDebug();
301     qDebug() << "Message3: " << QCA::arrayToHex(message3.toByteArray());
302     qDebug() << "Expecting: dfa66747de9ae63030ca32611497c827";
303     qDebug() << "AES-CMAC " << QCA::arrayToHex(cmacObject.process(message3).toByteArray());
304
305     cmacObject.clear();
306     QCA::SecureArray message4(message);
307     message4.resize(64);
308     qDebug();
309     qDebug() << "Message4: " << QCA::arrayToHex(message4.toByteArray());
310     qDebug() << "Expecting: 51f0bebf7e3b9d92fc49741779363cfe";
311     qDebug() << "AES-CMAC: " << QCA::arrayToHex(cmacObject.process(message4).toByteArray());
312 }
313
314 return 0;
315 }
316
```


12.2 base64test.cpp

The code below shows some simple operations on a `QCA::Base64` object, converting between `QCA::SecureArray` and `QString`.

```

1  /*
2  Copyright (C) 2004-2005 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26
27 #include <iostream>
28
29 int main(int argc, char **argv)
30 {
31     QCoreApplication(argc, argv);
32
33     // the Initializer object sets things up, and
34     // also does cleanup when it goes out of scope
35     QCA::Initializer init;
36
37     // we use the first argument as the data to encode
38     // if an argument is provided. Use "hello" if no argument
39     QByteArray arg; // empty array
40     arg.append((argc >= 2) ? argv[1] : "hello");
41
42     // create our object, which does encoding by default
43     // QCA::Base64 encoder(QCA::Encode); is equivalent
44     QCA::Base64 encoder;
45
46     // This does the actual conversion (encoding).
47     // You might prefer to use encoder.encode(); and have
48     // it return a QCA::SecureArray, depending on your needs
49     QString encoded = encoder.arrayToString(arg);
50
51     std::cout << arg.data() << " in base64 encoding is ";
52     std::cout << encoded.toLatin1().data() << std::endl;
53
54     // This time, we'll create an object to decode base64. We
55     // could also have reused the existing object, calling
56     // clear(); and setup(QCA::Decode); on it.
57     QCA::Base64 decoder(QCA::Decode);
58
59     // This time, we convert a QString into a QByteArray
60     QByteArray decoded = decoder.decodeString(encoded);
61

```

```
62         std::cout << encoded.toLatin1().data() << " decoded from base64 is ";
63         std::cout << decoded.toLatin1().data() << std::endl;
64
65         return 0;
66     }
67 }
```

12.3 certtest.cpp

This example shows how [QCA::Certificate](#) and [QCA::CertificateCollection](#) can be used. Note that the argument, if you provide it, must be a PEM encoded file collection.

```

1  /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2005 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23
24 #include <QtCrypto>
25
26 #include <QCoreApplication>
27
28 #include <iostream>
29
30 // dump out information about some part of the certificate
31 // we use this same approach for information about the subject
32 // of the certificate, and also about the issuer of the certificate
33 static void dumpCertificateInfo( QCA::CertificateInfo info)
34 {
35     std::cout << "  Organization: " << std::endl;
36
37     // Note that a single certificate can apply to more than one
38     // organisation. QCA::Certificate is a multimap, so when you
39     // ask for the values associated with a parameter, it returns
40     // a list.
41     QList<QString> orgInfoList = info.values(QCA::Organization);
42
43     // foreach() iterates over each value in the list, and we dump
44     // out each value. Note that is uncommon for a certificate to
45     // actually contain multiple values for a single parameter.
46     QString organization;
47     foreach( organization, orgInfoList ) {
48         std::cout << "      " << qPrintable(organization) << std::endl;
49     }
50
51     std::cout << "  Country: " << std::endl;
52     // As above, however this shows a more compact way to represent
53     // the iteration and output.
54     foreach( QString country, info.values(QCA::Country) ) {
55         std::cout << "      " << qPrintable(country) << std::endl;
56     }
57 }
58
59 // This is just a convenience routine
60 static void dumpSubjectInfo( QCA::CertificateInfo subject)
61 {

```

```
62     std::cout << "Subject: " << std::endl;
63
64     dumpCertificateInfo( subject );
65 }
66
67
68 // This is just a convenience routine
69 static void dumpIssuerInfo( QCA::CertificateInfo issuer)
70 {
71     std::cout << "Issuer: " << std::endl;
72
73     dumpCertificateInfo( issuer );
74 }
75
76
77 int main(int argc, char** argv)
78 {
79     // the Initializer object sets things up, and
80     // also does cleanup when it goes out of scope
81     QCA::Initializer init;
82
83     QCoreApplication app(argc, argv);
84
85     // We need to ensure that we have certificate handling support
86     if ( !QCA::isSupported( "cert" ) ) {
87         std::cout << "Sorry, no PKI certificate support" << std::endl;
88         return 1;
89     }
90
91     // We are going to work with a number of certificates, and a
92     // QList is a great template class for that
93     QList<QCA::Certificate> certlist;
94
95     // We do two different cases - if we provide an argument, it is taken
96     // as a filename to read the keys from. If there is no argument, we just
97     // read from the system store certificates.
98     if (argc >= 2) {
99         // we are going to read the certificates in using a single call
100         // which requires a CertificateCollection.
101         QCA::CertificateCollection filecerts;
102         // The conversion can be tested (although you don't have to) to find out if it
103         // worked.
104         QCA::ConvertResult importResult;
105         // This imports all the PEM encoded certificates from the file specified as the argument
106         // Note that you pass in a pointer to the result argument.
107         filecerts = QCA::CertificateCollection::fromFlatTextFile( argv[1], &importResult );
108         if ( QCA::ConvertGood == importResult ) {
109             std::cout << "Import succeeded" << std::endl;
110             // this turns the CertificateCollection into a QList of Certificate objects
111             certlist = filecerts.certificates();
112         }
113     } else {
114         // we have no arguments, so just use the system certificates
115         if ( !QCA::haveSystemStore() ) {
116             std::cout << "System certificates not available" << std::endl;
117             return 2;
118         }
119
120         // Similar to above, except we just want the system certificates
121         QCA::CertificateCollection systemcerts = QCA::systemStore();
122
123         // this turns the CertificateCollection into a QList of Certificate objects
124         certlist = systemcerts.certificates();
125     }
126
127     std::cout << "Number of certificates: " << certlist.count() << std::endl;
128 }
```

```
129     QCA::Certificate cert;
130     foreach (cert, certlist) {
131         std::cout << "Serial Number:";
132         // the serial number of the certificate is a QCA::BigInteger, but we can
133         // just convert it to a string, and then output it.
134         std::cout << qPrintable(cert.serialNumber().toString()) << std::endl;
135
136         // The subject information shows properties of who the certificate
137         // applies to. See the convenience routines above.
138         dumpSubjectInfo( cert.subjectInfo() );
139
140         // The issuer information shows properties of who the certificate
141         // was signed by. See the convenience routines above.
142         dumpIssuerInfo( cert.issuerInfo() );
143
144         // Test if the certificate can be used as a certificate authority
145         if ( cert.isCA() ) {
146             std::cout << "Is certificate authority" << std::endl;
147         } else {
148             std::cout << "Is not a certificate authority" << std::endl;
149         }
150
151         // Test if the certificate is self-signed.
152         if (cert.isSelfSigned() ) {
153             std::cout << "Self signed" << std::endl;
154         } else {
155             std::cout << "Is not self-signed!!!" << std::endl;
156         }
157
158         // Certificate are only valid between specific dates. We can get the dates
159         // (as a QDateTime) using a couple of calls
160         std::cout << "Valid from " << qPrintable(cert.notValidBefore().toString());
161         std::cout << ", until " << qPrintable(cert.notValidAfter().toString());
162         std::cout << std::endl;
163
164         // You can get the certificate in PEM encoding with a simple toPEM() call
165         std::cout << "PEM:" << std::endl;
166         std::cout << qPrintable(cert.toPEM());
167         std::cout << std::endl << std::endl;
168     }
169
170     return 0;
171 }
172
```

12.4 ciphertest.cpp

The code below shows the normal way to use the [QCA::Cipher](#) class.

```
1 /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2005-2006 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23 // QtCrypto has the declarations for all of QCA
24 #include <QtCrypto>
25 #include <stdio.h>
26
27 #include <QCoreApplication>
28
29 int main(int argc, char **argv)
30 {
31     QCoreApplication app(argc, argv);
32
33     // the Initializer object sets things up, and
34     // also does cleanup when it goes out of scope
35     QCA::Initializer init;
36
37     // we use the first argument if provided, or
38     // use "hello" if no arguments
39     QCA::SecureArray arg = (argc >= 2) ? argv[1] : "hello";
40
41     // AES128 testing
42     if(!QCA::isSupported("aes128-cbc-pkcs7"))
43         printf("AES128-CBC not supported!\n");
44     else {
45         // Create a random key - you'd probably use one from another
46         // source in a real application
47         QCA::SymmetricKey key(16);
48
49         // Create a random initialisation vector - you need this
50         // value to decrypt the resulting cipher text, but it
51         // need not be kept secret (unlike the key).
52         QCA::InitializationVector iv(16);
53
54         // create a 128 bit AES cipher object using Cipher Block Chaining (CBC) mode
55         QCA::Cipher cipher(QString("aes128"), QCA::Cipher::CBC,
56                             // use Default padding, which is equivalent to PKCS7 for CBC
57                             QCA::Cipher::DefaultPadding,
58                             // this object will encrypt
59                             QCA::Encode,
60                             key, iv);
61
62         // we use the cipher object to encrypt the argument we passed in
```

```

63         // the result of that is returned - note that if there is less than
64         // 16 bytes (1 block), then nothing will be returned - it is buffered
65         // update() can be called as many times as required.
66         QCA::SecureArray u = cipher.update(arg);
67
68         // We need to check if that update() call worked.
69         if (!cipher.ok()) {
70             printf("Update failed\n");
71         }
72         // output the results of that stage
73         printf("AES128 encryption of %s is [%s]\n",
74             arg.data(),
75             qPrintable(QCA::arrayToHex(u.toByteArray())) );
76
77
78         // Because we are using PKCS7 padding, we need to output the final (padded) block
79         // Note that we should always call final() even with no padding, to clean up
80         QCA::SecureArray f = cipher.final();
81
82         // Check if the final() call worked
83         if (!cipher.ok()) {
84             printf("Final failed\n");
85         }
86         // and output the resulting block. The ciphertext is the results of update()
87         // and the result of final()
88         printf("Final block for AES128 encryption is [0x%s]\n", qPrintable(QCA::arrayToHex(f.toByteArray())) );
89
90         // re-use the Cipher to decrypt. We need to use the same key and
91         // initialisation vector as in the encryption.
92         cipher.setup( QCA::Decode, key, iv );
93
94         // Build a single cipher text array. You could also call update() with
95         // each block as you receive it, if that is more useful.
96         QCA::SecureArray cipherText = u.append(f);
97
98         // take that cipher text, and decrypt it
99         QCA::SecureArray plainText = cipher.update(cipherText);
100
101         // check if the update() call worked
102         if (!cipher.ok()) {
103             printf("Update failed\n");
104         }
105
106         // output results
107         printf("Decryption using AES128 of [0x%s] is %s\n",
108             qPrintable(QCA::arrayToHex(cipherText.toByteArray())), plainText.data());
109
110         // Again we need to call final(), to get the last block (with its padding removed)
111         plainText = cipher.final();
112
113         // check if the final() call worked
114         if (!cipher.ok()) {
115             printf("Final failed\n");
116         }
117
118         // output results
119         printf("Final decryption block using AES128 is %s\n", plainText.data());
120         // instead of update() and final(), you can do the whole thing
121         // in one step, using process()
122         printf("One step decryption using AES128: %s\n",
123             QCA::SecureArray(cipher.process(cipherText)).data() );
124     }
125 }
126
127 return 0;
128 }
129

```

12.5 cmsexample.cpp

The code below shows the normal way to use the `QCA::CMS` class.

```
1 /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2005-2006 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23
24 #include <QtCrypto>
25
26 #include <QCoreApplication>
27 #include <QDebug>
28
29 int main(int argc, char** argv)
30 {
31     // the Initializer object sets things up, and
32     // also does cleanup when it goes out of scope
33     QCA::Initializer init;
34
35     QCoreApplication app(argc, argv);
36
37     // We need to ensure that we have certificate handling support
38     if ( !QCA::isSupported( "cert" ) ) {
39         qWarning() << "Sorry, no PKI certificate support";
40         return 1;
41     }
42
43     // Read in a public key cert
44     // you could also build this using the fromPEMFile() method
45     QCA::Certificate pubCert( "User.pem" );
46     if ( pubCert.isNull() ) {
47         qWarning() << "Sorry, could not import public key certificate";
48         return 1;
49     }
50     // We are building the certificate into a SecureMessageKey object, via a
51     // CertificateChain
52     QCA::SecureMessageKey secMsgKey;
53     QCA::CertificateChain chain;
54     chain += pubCert;
55     secMsgKey.setX509CertificateChain( chain );
56
57     // build up a SecureMessage object, based on our public key certificate
58     if ( !QCA::isSupported( "cms" ) ) {
59         qWarning() << "Sorry, no CMS support";
60         return 1;
61     }
62     QCA::CMS cms;
```



```

63     QCA::SecureMessage msg(&cms);
64     msg.setRecipient(secMsgKey);
65
66     // Some plain text - we use the first command line argument if provided
67     QByteArray plainText = (argc >= 2) ? argv[1] : "What do ya want for nuthin'";
68
69     // Now use the SecureMessage object to encrypt the plain text.
70     msg.startEncrypt();
71     msg.update(plainText);
72     msg.end();
73     // I think it is reasonable to wait for 1 second for this
74     msg.waitForFinished(1000);
75
76     // check to see if it worked
77     if(!msg.success())
78     {
79         qWarning() << "Error encrypting: " << msg.errorCode();
80         return 1;
81     }
82
83     // get the result
84     QByteArray cipherText = msg.read();
85     QCA::Base64 enc;
86     qDebug() << "" << plainText.data() << " encrypts to (in base 64): ";
87     qDebug() << enc.arrayToString( cipherText );
88     qDebug() << "Message uses" << msg.hashName() << "hashing algorithm";
89     qDebug();
90
91     // Show we can decrypt it with the private key
92
93     // Read in a private key
94     QCA::PrivateKey privKey;
95     QCA::ConvertResult convRes;
96     QCA::SecureArray passPhrase = "start";
97     privKey = QCA::PrivateKey::fromPEMFile( "Userkey.pem", passPhrase, &convRes );
98     if ( convRes != QCA::ConvertGood ) {
99         qWarning() << "Sorry, could not import Private Key";
100        return 1;
101    }
102
103    QCA::SecureMessageKey secMsgKey2;
104    // needed?
105    secMsgKey2.setX509CertificateChain( chain );
106    secMsgKey2.setX509PrivateKey(privKey);
107    QCA::SecureMessageKeyList privKeyList;
108    privKeyList += secMsgKey2;
109
110    // build up a SecureMessage object, based on the private key
111    // you could re-use the existing QCA::CMS object (cms), but
112    // this example simulates encryption and one end, and decryption
113    // at the other
114    QCA::CMS anotherCms;
115    anotherCms.setPrivateKeys( privKeyList );
116
117    QCA::SecureMessage msg2( &anotherCms );
118
119    msg2.startDecrypt();
120    msg2.update( cipherText );
121    msg2.end();
122
123    // I think it is reasonable to wait for 1 second for this
124    msg2.waitForFinished(1000);
125
126    // check to see if it worked
127    if(!msg2.success())
128    {
129        qWarning() << "Error encrypting: " << msg2.errorCode();

```

```
130         return 1;
131     }
132
133     QCA::SecureArray plainTextResult = msg2.read();
134
135     qDebug() << enc.arrayToString( cipherText )
136             << " (in base 64) decrypts to: "
137             << plainTextResult.data();
138
139     if (msg2.wasSigned()) {
140         qDebug() << "Message was signed at "
141                 << msg2.signer().timestamp();
142     } else {
143         qDebug() << "Message was not signed";
144     }
145
146     qDebug() << "Message used" << msg2.hashName() << "hashing algorithm";
147
148     qDebug();
149
150     // Now we want to try a signature
151     QByteArray text("Got your message");
152
153     // Re-use the CMS and SecureMessageKeyList objects from the decrypt...
154     QCA::SecureMessage signing( &anotherCms );
155     signing.setSigners(privKeyList);
156
157     signing.startSign(QCA::SecureMessage::Detached);
158     signing.update(text);
159     signing.end();
160
161     // I think it is reasonable to wait for 1 second for this
162     signing.waitForFinished(1000);
163
164     // check to see if it worked
165     if(!signing.success())
166     {
167         qWarning() << "Error signing: " << signing.errorCode();
168         return 1;
169     }
170
171     // get the result
172     QByteArray signature = signing.signature();
173
174     qDebug() << "'" << text.data() << "'", signature (converted to base 64), is: ";
175     qDebug() << enc.arrayToString( signature );
176     qDebug() << "Message uses" << signing.hashName() << "hashing algorithm";
177     qDebug();
178
179
180     // Now we go back to the first CMS, and re-use that.
181     QCA::SecureMessage verifying( &cms );
182
183     // You have to pass the signature to startVerify(),
184     // and the message to update()
185     verifying.startVerify(signature);
186     verifying.update(text);
187     verifying.end();
188
189     verifying.waitForFinished(1000);
190
191     // check to see if it worked
192     if(!verifying.success())
193     {
194         qWarning() << "Error verifying: " << verifying.errorCode();
195         return 1;
196     }
197 }
```

```
197
198     QCA::SecureMessageSignature sign;
199     sign = verifying.signer();
200     // todo: dump some data out about the signer
201
202     if(verifying.verifySuccess())
203     {
204         qDebug() << "Message verified";
205     } else {
206         qDebug() << "Message failed to verify:" << verifying.errorCode();
207     }
208
209     return 0;
210 }
211
```

12.6 eventhandlerdemo.cpp

The code below shows to implement a client side handler for password / passphrase / PIN and token requests from [QCA](#) and any associated providers.

```

1  /*
2  Copyright (C) 2007 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26
27 #include <iostream>
28
29 class ClientPassphraseHandler: public QObject
30 {
31     Q_OBJECT
32 public:
33     ClientPassphraseHandler(QObject *parent = 0) : QObject( parent )
34     {
35         // When the PasswordAsker or TokenAsker needs to interact
36         // with the user, it raises a signal. We connect that to a
37         // local slot to get the required information.
38         connect( &m_handler, SIGNAL( eventReady(int, const QCA::Event &) ),
39                 SLOT( my_eventReady(int, const QCA::Event &) ) );
40
41         // Now that we are set up, we can start the EventHandler. Nothing
42         // will happen if you don't call this method.
43         m_handler.start();
44     }
45
46 private slots:
47     // This slot gets called when the provider needs a token inserted,
48     // or to get a passphrase / password / PIN.
49     void my_eventReady(int id, const QCA::Event &event)
50     {
51         // We can sanity check the event
52         if ( event.isNull() ) {
53             return;
54         }
55
56         // Events can be associated with a a keystore or a file/bytearray
57         // You can tell which by looking at the Source
58         if ( event.source() == QCA::Event::KeyStore ) {
59             std::cout << "Event is associated with a key store operation" << std::endl;
60         } else if ( event.source() == QCA::Event::Data ) {
61             std::cout << "Event is associated with a file or some other data" << std::endl;
62         }
63     }
64 }

```

```

65         // if the event comes from a file type operation, you can get the
66         // name / label using fileName()
67         std::cout << "    Filename: " << qPrintable( event.fileName() ) << std::endl;
68     } else {
69         std::cout << "Unexpected Source for Event" << std::endl;
70     }
71
72     // There are different kinds of events.
73     if ( event.type() == QCA::Event::Token ) {
74         // You would typically ask the user to insert the token here
75         std::cout << "Request for token" << std::endl;
76         // we just fake it for this demo.
77         m_handler.tokenOkay( id );
78         // you could use m_handler.reject( id ) to refuse the token request
79
80     } else if ( event.type() == QCA::Event::Password ) {
81         std::cout << "Request for password, passphrase or PIN" << std::endl;
82         // and within the Password type, we have a few different styles.
83         if ( event.passwordStyle() == QCA::Event::StylePassword ) {
84             std::cout << "    [Password request]" << std::endl;
85         } else if ( event.passwordStyle() == QCA::Event::StylePassphrase ) {
86             std::cout << "    [Passphrase request]" << std::endl;
87         } else if ( event.passwordStyle() == QCA::Event::StylePIN ) {
88             std::cout << "    [PIN request]" << std::endl;
89         } else {
90             std::cout << "    [unexpected request style]" << std::endl;
91         }
92         // You would typically request the password/PIN/passphrase.
93         // again, we just fake it.
94         m_handler.submitPassword( id, QCA::SecureArray( "hello" ) );
95
96     } else {
97         std::cout << "Unexpected event type" << std::endl;
98     }
99 }
100 private:
101     QCA::EventHandler m_handler;
102
103 };
104
105 void asker_procedure();
106
107 class AskerThread : public QThread
108 {
109     Q_OBJECT
110 protected:
111     virtual void run()
112     {
113         asker_procedure();
114     }
115 };
116
117 int main(int argc, char **argv)
118 {
119     QCoreApplication exampleApp(argc, argv);
120
121     // the Initializer object sets things up, and
122     // also does cleanup when it goes out of scope
123     QCA::Initializer init;
124
125     ClientPassphraseHandler cph;
126
127     // handler and asker cannot occur in the same thread
128     AskerThread askerThread;
129     QObject::connect(&askerThread, SIGNAL(finished()), &exampleApp, SLOT(quit()));
130     askerThread.start();
131

```

```
132     exampleApp.exec();
133     return 0;
134 }
135
136 void asker_procedure()
137 {
138     QCA::PasswordAsker pwAsker;
139
140     pwAsker.ask( QCA::Event::StylePassword, "foo.tmp", 0 );
141
142     pwAsker.waitForResponse();
143
144     std::cout << "Password was: " << pwAsker.password().toByteArray().data() << std::endl;
145
146     std::cout << std::endl << "Now do token:" << std::endl;
147
148     QCA::TokenAsker tokenAsker;
149
150     tokenAsker.ask( QCA::KeyStoreInfo( QCA::KeyStore::SmartCard, "Token Id", "Token Name" ), QCA::KeyS
151
152     tokenAsker.waitForResponse();
153
154     if ( tokenAsker.accepted() ) {
155         std::cout << "Token was accepted" << std::endl;
156     } else {
157         std::cout << "Token was not accepted" << std::endl;
158     }
159 }
160
161 #include "eventhandlerdemo.moc"
```

12.7 hashtest.cpp

The code below shows how to use the [QCA::Hash](#) class

```

1  /*
2   Copyright (C) 2004 Brad Hards <bradh@frogmouth.net>
3
4   Permission is hereby granted, free of charge, to any person obtaining a copy
5   of this software and associated documentation files (the "Software"), to deal
6   in the Software without restriction, including without limitation the rights
7   to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8   copies of the Software, and to permit persons to whom the Software is
9   furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto/QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26
27 #include <stdio.h>
28
29 int main(int argc, char **argv)
30 {
31     // the Initializer object sets things up, and
32     // also does cleanup when it goes out of scope
33     QCA::Initializer init;
34
35     QCoreApplication app(argc, argv);
36
37     // we use the first argument if provided, or
38     // use "hello" if no arguments
39     QCA::SecureArray arg = (argc >= 2) ? argv[1] : "hello";
40
41     // must always check that an algorithm is supported before using it
42     if( !QCA::isSupported("sha1") )
43         printf("SHA1 not supported!\n");
44     else {
45         // this shows the "all in one" approach
46         QString result = QCA::Hash("sha1").hashToString(arg);
47         printf("sha1(\"%s\") = [%s]\n", arg.data(), result.toAscii().data());
48     }
49
50     // must always check that an algorithm is supported before using it
51     if( !QCA::isSupported("md5") )
52         printf("MD5 not supported!\n");
53     else {
54         // this shows the incremental approach. Naturally
55         // for this simple job, we could use the "all in one"
56         // approach - this is an example, after all :-)
57         QCA::SecureArray part1(arg.toByteArray().left(3)); // three chars - "hel"
58         QCA::SecureArray part2(arg.toByteArray().mid(3)); // the rest - "lo"
59
60         // create the required object.
61         QCA::Hash hashObject("md5");
62         // we split it into two parts to show incremental update

```

```
63         hashObject.update(part1);
64         hashObject.update(part2);
65         // no more updates after calling final.
66         QCA::SecureArray resultArray = hashObject.final();
67         // convert the result into printable hexadecimal.
68         QString result = QCA::arrayToHex(resultArray.toByteArray());
69         printf("md5(\"%s\") = [%s]\n", arg.data(), result.toAscii().data());
70     }
71
72     return 0;
73 }
74
```


12.8 hextest.cpp

The code below shows some simple operations on a [QCA::Hex](#) object, converting between [QCA::SecureArray](#) and [QString](#).

```

1  /*
2  Copyright (C) 2005 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26
27 #include <iostream>
28
29 int main(int argc, char **argv)
30 {
31     QCoreApplication app(argc, argv);
32
33     // the Initializer object sets things up, and
34     // also does cleanup when it goes out of scope
35     QCA::Initializer init;
36
37     // we use the first argument as the data to encode / decode
38     // if an argument is provided. Use "hello" if no argument
39     QByteArray arg; // empty array
40     arg.append((argc >= 2) ? argv[1] : "hello");
41
42     // create our object, which does encoding by default
43     // QCA::Hex encoder(QCA::Encode); is equivalent
44     QCA::Hex encoder;
45
46     // You might prefer to use encoder.encode(); and have
47     // it return a QCA::SecureArray, depending on your needs
48     QString encoded = encoder.arrayToString(arg);
49
50     std::cout << arg.data() << " in hex encoding is ";
51     std::cout << encoded.toLatin1().data() << std::endl;
52
53     // This time, we'll create an object to decode hexadecimal.
54     // We could also have reused the existing object, calling
55     // clear(); and setup(QCA::Decode); on it.
56     QCA::Hex decoder(QCA::Decode);
57
58     // This time, we convert a QString into a QByteArray
59     QByteArray decoded = decoder.decodeString(encoded);
60
61     std::cout << decoded.toLatin1().data() << " decoded from hex is ";

```

```
62         std::cout << decoded.toLatin1().data() << std::endl;
63
64         return 0;
65     }
66
```

12.9 keyloader.cpp

The code below shows how to load a private key from a PEM format file, including handling any requirement for a passphrase. This is done using the [QCA::KeyLoader](#) class.

```

1  /*
2  Copyright (C) 2007 Justin Karneges <justin@affinix.com>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26 #include <QTimer>
27
28 class PassphraseHandler: public QObject
29 {
30     Q_OBJECT
31 public:
32     QCA::EventHandler handler;
33
34     PassphraseHandler(QObject *parent = 0) : QObject(parent)
35     {
36         connect(&handler, SIGNAL(eventReady(int, const QCA::Event &)),
37             SLOT(eh_eventReady(int, const QCA::Event &)));
38         handler.start();
39     }
40
41 private slots:
42     void eh_eventReady(int id, const QCA::Event &event)
43     {
44         if(event.type() == QCA::Event::Password)
45         {
46             QCA::SecureArray pass;
47             QCA::ConsolePrompt prompt;
48             prompt.getHidden("Passphrase");
49             prompt.waitForFinished();
50             pass = prompt.result();
51             handler.submitPassword(id, pass);
52         }
53         else
54             handler.reject(id);
55     }
56 };
57
58 class App : public QObject
59 {
60     Q_OBJECT
61 public:

```

```
62     QCA::KeyLoader keyLoader;
63     QString str;
64
65     App()
66     {
67         connect(&keyLoader, SIGNAL(finished()), SLOT(kl_finished()));
68     }
69
70 public slots:
71     void start()
72     {
73         keyLoader.loadPrivateKeyFromPEMFile(str);
74     }
75
76 signals:
77     void quit();
78
79 private slots:
80     void kl_finished()
81     {
82         if(keyLoader.convertResult() == QCA::ConvertGood)
83         {
84             QCA::PrivateKey key = keyLoader.privateKey();
85             printf("Loaded successfully. Bits: %d\n", key.bitSize());
86         }
87         else
88             printf("Unable to load.\n");
89
90         emit quit();
91     }
92 };
93
94 int main(int argc, char **argv)
95 {
96     QCA::Initializer init;
97     QApplication qapp(argc, argv);
98
99     if(argc < 2)
100     {
101         printf("usage: keyloader [privatekey.pem]\n");
102         return 0;
103     }
104
105     PassphraseHandler passphraseHandler;
106     App app;
107     app.str = argv[1];
108     QObject::connect(&app, SIGNAL(quit()), &qapp, SLOT(quit()));
109     QTimer::singleShot(0, &app, SLOT(start()));
110     qapp.exec();
111     return 0;
112 }
113
114 #include "keyloader.moc"
```

12.10 mactest.cpp

The code below shows how to use the [QCA::MessageAuthenticationCode](#) class

```

1  /*
2   Copyright (C) 2004, 2006 Brad Hards <bradh@frogmouth.net>
3
4   Permission is hereby granted, free of charge, to any person obtaining a copy
5   of this software and associated documentation files (the "Software"), to deal
6   in the Software without restriction, including without limitation the rights
7   to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8   copies of the Software, and to permit persons to whom the Software is
9   furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20  */
21
22  // QtCrypto has the declarations for all of QCA
23  #include <QtCrypto>
24
25  #include <QCoreApplication>
26  #include <QDebug>
27
28  // needed for printf
29  #include <stdio.h>
30
31  int main(int argc, char **argv)
32  {
33      QCoreApplication app(argc, argv);
34
35      qDebug() << "This example shows hashed MAC";
36
37      // the_INITIALIZER object sets things up, and
38      // also does cleanup when it goes out of scope
39      QCA::Initializer init;
40
41      // we use the first argument as the data to authenticate
42      // if an argument is provided. Use "hello" if no argument
43      QByteArray arg = (argc >= 2) ? argv[1] : "hello";
44
45      // we use the second argument as the key to authenticate
46      // with, if two arguments are provided. Use "secret" as
47      // the key if less than two arguments.
48      QCA::SecureArray key((argc >= 3) ? argv[2] : "secret");
49
50      // must always check that an algorithm is supported before using it
51      if( !QCA::isSupported("hmac(sha1)") ) {
52          printf("HMAC(SHA1) not supported!\n");
53      } else {
54          // create the required object using HMAC with SHA-1, and an
55          // empty key.
56          QCA::MessageAuthenticationCode hmacObject( "hmac(sha1)", QCA::SecureArray() );
57
58          // create the key
59          QCA::SymmetricKey keyObject(key);
60
61          // set the HMAC object to use the key
62          hmacObject.setup(key);

```

```
63         // that could also have been done in the
64         // QCA::MessageAuthenticationCode constructor
65
66         // we split it into two parts to show incremental update
67         QCA::SecureArray part1(arg.left(3)); // three chars - "hel"
68         QCA::SecureArray part2(arg.mid(3)); // the rest - "lo"
69         hmacObject.update(part1);
70         hmacObject.update(part2);
71
72         // no more updates after calling final.
73         QCA::SecureArray resultArray = hmacObject.final();
74
75         // convert the result into printable hexadecimal.
76         QString result = QCA::arrayToHex(resultArray.toByteArray());
77         printf("HMAC(SHA1) of \"%s\" with \"%s\" = [%s]\n", arg.data(), key.data(), result.toLa
78     }
79
80     return 0;
81 }
82
```

12.11 main.cpp

The code below shows how to use Cryptographic Message Syntax (CMS) in a GUI application.

```
1 /*
2  Copyright (C) 2007 Justin Karneges <justin@affinix.com>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 #include "tlssocket.h"
23
24 #include <QCoreApplication>
25
26 int main(int argc, char **argv)
27 {
28     QCA::Initializer init;
29     QCoreApplication qapp(argc, argv);
30
31     TLSSocket socket;
32     socket.connectToHostEncrypted("www.paypal.com", 443);
33     socket.write("GET / HTTP/1.0\r\n\r\n");
34     while(socket.waitForReadyRead())
35         printf("%s", socket.readAll().data());
36
37     return 0;
38 }
```

12.12 md5crypt.cpp

The code below shows how to calculate an md5crypt based password. This code is compatible with the glibc code.

```
1 /*
2  Copyright (C) 2007 Carlo Todeschini - Metarete s.r.l. <info@metarete.it>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 /*
23  Algorithm inspired by Vladimir Silva's "Secure Java apps on Linux using
24  MD5 crypt" article
25  (http://www-128.ibm.com/developerworks/linux/library/l-md5crypt/)
26 */
27
28 #include <QtCrypto>
29 #include <QCoreApplication>
30 #include <QtDebug>
31 #include <stdio.h>
32
33 QString to64 ( long v , int size )
34 {
35
36     // Character set of the encrypted password: A-Za-z0-9./
37     QString itoa64 = ".0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
38     QString result;
39
40     while ( --size >= 0 )
41     {
42         result.append ( itoa64.at ( ( int )( v & 0x3f ) ) );
43         v = v >> 6;
44     }
45
46     return result;
47
48 }
49
50
51 int byte2unsigned ( int byteValue )
52 {
53
54     int integerToReturn;
55     integerToReturn = (int) byteValue & 0xff;
56     return integerToReturn;
57
58 }
59
60 QString qca_md5crypt( const QCA::SecureArray &password, const QCA::SecureArray &salt )
61 {
```



```

62     QCA::SecureArray finalState, magic_string = "$1$";
63
64     // The md5crypt algorithm uses two separate hashes
65     QCA::Hash hash1( "md5" );
66     QCA::Hash hash2( "md5" );
67
68     // MD5 Hash #1: pwd, magic string and salt
69     hash1.update ( password );
70     hash1.update ( magic_string );
71     hash1.update ( salt );
72
73     // MD5 Hash #2: password, salt, password
74     hash2.update ( password );
75     hash2.update ( salt );
76     hash2.update ( password );
77
78     finalState = hash2.final();
79
80     // Two sets of transformations based on the length of the password
81     for ( int i = password.size() ; i > 0 ; i -= 16 )
82     {
83         // Update hash1 from offset value (i > 16 ? 16 : i)
84         hash1.update( finalState.toByteArray().left(i > 16 ? 16 : i));
85     }
86
87     // Clear array bits
88     finalState.fill( 0 );
89
90     for ( int i = password.size() ; i != 0 ; i = i >> 1 )
91     {
92         if ( ( i & 1 ) != 0 )
93         {
94             hash1.update( finalState.toByteArray().left ( 1 ) );
95         }
96         else
97         {
98             hash1.update( password.toByteArray().left ( 1 ) );
99         }
100     }
101
102     finalState = hash1.final();
103
104     // Now build a 1000 entry dictionary...
105     for ( int i = 0 ; i < 1000 ; i++ )
106     {
107
108         hash2.clear();
109
110         if ((i & 1) != 0)
111         {
112             hash2.update ( password );
113         }
114         else
115         {
116             hash2.update ( finalState.toByteArray().left( 16 ) );
117         }
118
119         if ((i % 3) != 0)
120         {
121             hash2.update ( salt );
122         }
123
124         if ((i % 7) != 0)
125         {
126             hash2.update ( password );
127         }
128

```

```
129         if ((i & 1) != 0)
130         {
131             hash2.update ( finalState.toByteArray().left( 16 ) );
132         }
133         else
134         {
135             hash2.update ( password );
136         }
137
138         finalState = hash2.final();
139     }
140
141     // Create an output string
142     // Salt is part of the encoded password ($1$<string>$)
143     QString encodedString;
144
145     encodedString.append ( magic_string.toByteArray() );
146     encodedString.append ( salt.toByteArray() );
147     encodedString.append ( "$" );
148
149     long l;
150
151     l = ( byte2unsigned (finalState.toByteArray().at(0) ) << 16 |
152          ( byte2unsigned (finalState.toByteArray().at(6)) ) << 8 |
153          byte2unsigned (finalState.toByteArray().at(12)) );
154     encodedString.append ( to64 (l, 4) );
155
156     l = ( byte2unsigned (finalState.toByteArray().at(1)) << 16 |
157          ( byte2unsigned (finalState.toByteArray().at(7)) ) << 8 |
158          byte2unsigned (finalState.toByteArray().at(13)) );
159     encodedString.append ( to64 (l, 4) );
160
161     l = ( byte2unsigned (finalState.toByteArray().at(2)) << 16 |
162          ( byte2unsigned (finalState.toByteArray().at(8)) ) << 8 |
163          byte2unsigned (finalState.toByteArray().at(14)) );
164     encodedString.append ( to64 (l, 4) );
165
166     l = ( byte2unsigned (finalState.toByteArray().at(3)) << 16 |
167          ( byte2unsigned (finalState.toByteArray().at(9)) ) << 8 |
168          byte2unsigned (finalState.toByteArray().at(15)) );
169     encodedString.append ( to64 (l, 4) );
170
171     l = ( byte2unsigned (finalState.toByteArray().at(4)) << 16 |
172          ( byte2unsigned (finalState.toByteArray().at(10)) ) << 8 |
173          byte2unsigned (finalState.toByteArray().at(5)) );
174     encodedString.append ( to64 (l, 4) );
175
176     l = byte2unsigned (finalState.toByteArray().at(11));
177     encodedString.append ( to64 (l, 2) );
178
179     return encodedString;
180 }
181
182 int main(int argc, char **argv)
183 {
184
185     // the_INITIALIZER object sets things up, and
186     // also does cleanup when it goes out of scope
187     QCA::Initializer init;
188
189     QCA::SecureArray password, salt;
190
191     QCoreApplication app ( argc, argv );
192
193     if ( argc < 3 )
194     {
195         printf ( "Usage: %s password salt (salt without $1$)\n" , argv[0] );
```

```
196         return 1;
197     }
198
199     password.append( argv[1] );
200
201     salt.append( argv[2] );
202
203     // must always check that an algorithm is supported before using it
204     if( !QCA::isSupported( "md5" ) )
205         printf ( "MD5 hash not supported!\n" );
206     else
207     {
208         QString result = qca_md5crypt( password, salt );
209
210         printf ( "md5crypt      [ %s , %s ] = '%s'\n" , password.data(), salt.data() , qPrintable(result) );
211
212         // this is equivalent if you have GNU libc 2.0
213         // printf( "GNU md5crypt [ %s , %s ] = '%s'\n", password.data(), salt.data(), crypt( password.data(), salt.data() ) );
214     }
215
216     return 0;
217 }
218
219
220
221
```

12.13 providertest.cpp

The code below shows some simple operations on a [QCA::Provider](#) object, including use of iterators and some member functions.

```

1  /*
2  Copyright (C) 2004 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24 #include <QCoreApplication>
25
26 #include <iostream>
27 #include <qstringlist.h>
28
29 int main(int argc, char **argv)
30 {
31     // the Initializer object sets things up, and
32     // also does cleanup when it goes out of scope
33     QCA::Initializer init;
34
35     QCoreApplication app(argc, argv);
36
37     // get all the available providers loaded.
38     // you don't normally need this (because you test using isSupported())
39     // but this is a special case.
40     QCA::scanForPlugins();
41
42     // this gives us all the plugin providers as a list
43     QCA::ProviderList qcaProviders = QCA::providers();
44     for (int i = 0; i < qcaProviders.size(); ++i) {
45         // each provider has a name, which we can display
46         std::cout << qcaProviders[i]->name().toLatin1().data() << ": ";
47         // ... and also a list of features
48         QStringList capabilities = qcaProviders[i]->features();
49         // we turn the string list back into a single string,
50         // and display it as well
51         std::cout << capabilities.join(", ").toLatin1().data() << std::endl;
52     }
53
54     // Note that the default provider isn't included in
55     // the result of QCA::providers()
56     std::cout << "default: ";
57     // However it is still possible to get the features
58     // supported by the default provider
59     QStringList capabilities = QCA::defaultFeatures();
60     std::cout << capabilities.join(", ").toLatin1().data() << std::endl;
61     return 0;

```

```
62 }  
63
```

12.14 publickeyexample.cpp

The code below shows how to do public key encryption, decryption, signing and verification.

```
1 /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2005 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23
24 #include <QtCrypto>
25
26 #include <QCoreApplication>
27
28 #include <iostream>
29
30
31 int main(int argc, char** argv)
32 {
33     // the Initializer object sets things up, and
34     // also does cleanup when it goes out of scope
35     QCA::Initializer init;
36
37     QCoreApplication app(argc, argv);
38
39     // We need to ensure that we have certificate handling support
40     if ( !QCA::isSupported( "cert" ) ) {
41         std::cout << "Sorry, no PKI certificate support" << std::endl;
42         return 1;
43     }
44
45     // Read in a private key
46     QCA::PrivateKey privKey;
47     QCA::ConvertResult convRes;
48     QCA::SecureArray passPhrase = "start";
49     privKey = QCA::PrivateKey::fromPEMFile( "Userkey.pem", passPhrase, &convRes );
50     if ( convRes != QCA::ConvertGood ) {
51         std::cout << "Sorry, could not import Private Key" << std::endl;
52         return 1;
53     }
54
55     // Read in a matching public key cert
56     // you could also build this using the fromPEMFile() method
57     QCA::Certificate pubCert( "User.pem" );
58     if ( pubCert.isNull() ) {
59         std::cout << "Sorry, could not import public key certificate" << std::endl;
60         return 1;
61     }
62     // We are building the certificate into a SecureMessageKey object, via a
```

```

63     // CertificateChain
64     QCA::SecureMessageKey secMsgKey;
65     QCA::CertificateChain chain;
66     chain += pubCert;
67     secMsgKey.setX509CertificateChain( chain );
68
69     // build up a SecureMessage object, based on our public key certificate
70     QCA::CMS cms;
71     QCA::SecureMessage msg(&cms);
72     msg.setRecipient(secMsgKey);
73
74     // Some plain text - we use the first command line argument if provided
75     QByteArray plainText = (argc >= 2) ? argv[1] : "What do ya want for nuthin'";
76
77     // Now use the SecureMessage object to encrypt the plain text.
78     msg.startEncrypt();
79     msg.update(plainText);
80     msg.end();
81     // I think it is reasonable to wait for 1 second for this
82     msg.waitForFinished(1000);
83
84     // check to see if it worked
85     if(!msg.success())
86     {
87         std::cout << "Error encrypting: " << msg.errorCode() << std::endl;
88         return 1;
89     }
90
91     // get the result
92     QCA::SecureArray cipherText = msg.read();
93     QCA::Base64 enc;
94     std::cout << plainText.data() << " encrypts to (in base 64): ";
95     std::cout << qPrintable( enc.arrayToString( cipherText ) ) << std::endl;
96
97     // Show we can decrypt it with the private key
98     if ( !privKey.canDecrypt() ) {
99         std::cout << "Private key cannot be used to decrypt" << std::endl;
100         return 1;
101     }
102     QCA::SecureArray plainTextResult;
103     if ( 0 == privKey.decrypt(cipherText, &plainTextResult, QCA::EME_PKCS1_OAEP ) ) {
104         std::cout << "Decryption process failed" << std::endl;
105         return 1;
106     }
107
108     std::cout << qPrintable( enc.arrayToString( cipherText ) );
109     std::cout << " (in base 64) decrypts to: ";
110     std::cout << plainTextResult.data() << std::endl;
111
112     return 0;
113 }
114

```

12.15 randomtest.cpp

The code below shows the normal way to use the [QCA::Random](#) class.

```
1 /*
2  Copyright (C) 2004, 2006 Brad Hards <bradh@frogmouth.net>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 // QtCrypto has the declarations for all of QCA
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26 #include <QDebug>
27
28 #include <iostream>
29
30 int main(int argc, char **argv)
31 {
32     QCoreApplication app(argc, argv);
33
34     qDebug() << "This example generates random numbers";
35
36     // the Initializer object sets things up, and
37     // also does cleanup when it goes out of scope
38     QCA::Initializer init;
39
40     int randInt;
41     // This is the standard way to generate a random integer.
42     randInt = QCA::Random::randomInt();
43     qDebug() << "A random number: " << randInt;
44
45     // If you wanted a random character (octet), you could
46     // use something like:
47     unsigned char randChar;
48     randChar = QCA::Random::randomChar();
49     // It might not be printable, so this may not produce output
50     std::cout << "A random character: " << randChar << std::endl;
51
52     QCA::SecureArray tenBytes(10);
53     // If you need more random values, you may want to
54     // get an array, as shown below.
55     tenBytes = QCA::Random::randomArray(10);
56
57     // To make this viewable, we convert to hexadecimal.
58     std::cout << "A random 10 byte array (in hex): ";
59     std::cout << QCA::Hex().arrayToString(tenBytes).toAscii().data() << std::endl;
60
61     // Under some circumstances, you may want to create a
62     // Random object, rather than a static public member function.
```



```
63         // This isn't normally the easiest way, but it does work
64         QCA::Random myRandomObject;
65         randChar = myRandomObject.nextByte();
66         tenBytes = myRandomObject.nextBytes(10);
67         return 0;
68     }
69 }
```

12.16 rsatest.cpp

The code below shows some of the capabilities for how to use RSA. This example also shows how to export and import a key to a file, using PEM encoding.

```
1 /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2005 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23 #include <QtCrypto>
24 #include <QCoreApplication>
25
26 #include <iostream>
27
28 int main(int argc, char **argv)
29 {
30     // The Initializer object sets things up, and also
31     // does cleanup when it goes out of scope
32     QCA::Initializer init;
33
34     QCoreApplication app(argc, argv);
35
36     // we use the first argument if provided, or
37     // use "hello" if no arguments
38     QCA::SecureArray arg = (argc >= 2) ? argv[1] : "hello";
39
40     // We demonstrate PEM usage here, so we need to test for
41     // supportedIOTypes, not just supportedTypes
42     if(!QCA::isSupported("pkey") ||
43         !QCA::PKey::supportedIOTypes().contains(QCA::PKey::RSA))
44         printf("RSA not supported!\n");
45     else {
46         // When creating a public / private key pair, you make the
47         // private key, and then extract the public key component from it
48         // Using RSA is very common, however DSA can provide equivalent
49         // signature/verification. This example applies to DSA to the
50         // extent that the operations work on that key type.
51
52         // QCA provides KeyGenerator as a convenient source of new keys,
53         // however you could also import an existing key instead.
54         QCA::PrivateKey seckey = QCA::KeyGenerator().createRSA(1024);
55         if(seckey.isNull()) {
56             std::cout << "Failed to make private RSA key" << std::endl;
57             return 1;
58         }
59
60         QCA::PublicKey pubkey = seckey.toPublicKey();
61     }
```

```

62         // check if the key can encrypt
63         if(!pubkey.canEncrypt()) {
64             std::cout << "Error: this kind of key cannot encrypt" << std::endl;
65             return 1;
66         }
67
68         // encrypt some data - note that only the public key is required
69         // you must also choose the algorithm to be used
70         QCA::SecureArray result = pubkey.encrypt(arg, QCA::EME_PKCS1_OAEP);
71         if(result.isEmpty()) {
72             std::cout << "Error encrypting" << std::endl;
73             return 1;
74         }
75
76         // output the encrypted data
77         QString rstr = QCA::arrayToHex(result.toByteArray());
78         std::cout << "\"" << arg.data() << "\" encrypted with RSA is \"";
79         std::cout << qPrintable(rstr) << "\"" << std::endl;
80
81         // save the private key - in a real example, make sure this goes
82         // somewhere secure and has a good pass phrase
83         // You can use the same technique with the public key too.
84         QCA::SecureArray passPhrase = "pass phrase";
85         seckey.toPEMFile("keyprivate.pem", passPhrase);
86
87         // Read that key back in, checking if the read succeeded
88         QCA::ConvertResult conversionResult;
89         QCA::PrivateKey privateKey = QCA::PrivateKey::fromPEMFile( "keyprivate.pem",
90                                                                    passPhrase,
91                                                                    &conversionResult);
92         if (! (QCA::ConvertGood == conversionResult) ) {
93             std::cout << "Private key read failed" << std::endl;
94         }
95
96         // now decrypt that encrypted data using the private key that
97         // we read in. The algorithm is the same.
98         QCA::SecureArray decrypt;
99         if(0 == privateKey.decrypt(result, &decrypt, QCA::EME_PKCS1_OAEP)) {
100             printf("Error decrypting.\n");
101             return 1;
102         }
103
104         // output the resulting decrypted string
105         std::cout << "\"" << qPrintable(rstr) << "\" decrypted with RSA is \"";
106         std::cout << decrypt.data() << "\"" << std::endl;
107
108
109         // Some private keys can also be used for producing signatures
110         if(!privateKey.canSign()) {
111             std::cout << "Error: this kind of key cannot sign" << std::endl;
112             return 1;
113         }
114         privateKey.startSign( QCA::EMSA3_MD5 );
115         privateKey.update( arg ); // just reuse the same message
116         QByteArray argSig = privateKey.signature();
117
118         // instead of using the startSign(), update(), signature() calls,
119         // you may be better doing the whole thing in one go, using the
120         // signMessage call. Of course you need the whole message in one
121         // hit, which may or may not be a problem
122
123         // output the resulting signature
124         rstr = QCA::arrayToHex(argSig);
125         std::cout << "Signature for \"" << arg.data() << "\" using RSA, is ";
126         std::cout << "\"" << qPrintable( rstr ) << "\"" << std::endl;
127
128         // to check a signature, we must check that the key is

```

```
129         // appropriate
130         if(pubkey.canVerify()) {
131             pubkey.startVerify( QCA::EMSA3_MD5 );
132             pubkey.update( arg );
133             if ( pubkey.validSignature( argSig ) ) {
134                 std::cout << "Signature is valid" << std::endl;
135             } else {
136                 std::cout << "Bad signature" << std::endl;
137             }
138         }
139
140         // We can also do the verification in a single step if we
141         // have all the message
142         if ( pubkey.canVerify() &&
143             pubkey.verifyMessage( arg, argSig, QCA::EMSA3_MD5 ) ) {
144             std::cout << "Signature is valid" << std::endl;
145         } else {
146             std::cout << "Signature could not be verified" << std::endl;
147         }
148     }
149 }
150
151 return 0;
152 }
153
```

12.17 saslservertest.cpp

The code below shows how to create a SASL server.

```

1  /*
2  Copyright (C) 2003-2006 Justin Karneges <justin@affinix.com>, Michail Pishchagin
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20  */
21
22  #include <QCoreApplication>
23  #include <QTimer>
24  #include <QTcpSocket>
25  #include <QTcpServer>
26  #include <stdio.h>
27
28  #ifdef Q_OS_UNIX
29  #include <unistd.h>
30  #endif
31
32  // QtCrypto has the declarations for all of QCA
33  #include <QtCrypto>
34
35  #define PROTO_NAME "foo"
36  #define PROTO_PORT 8001
37
38  class ServerTest : public QTcpServer
39  {
40      Q_OBJECT
41  public:
42      ServerTest(const QString &_str, int _port) : port(_port)
43      {
44          sock = 0;
45          sasl = 0;
46
47          connect(this, SIGNAL(newConnection()), SLOT(serv_newConnection()));
48          realm.clear();
49          str = _str;
50      }
51
52      ~ServerTest()
53      {
54          delete sock;
55          delete sasl;
56      }
57
58      void start()
59      {
60          if(!listen(QHostAddress::Any, port)) {
61              printf("Error binding to port %d!\n", port);
62              QTimer::singleShot(0, this, SIGNAL(quit()));

```

```

63             return;
64         }
65         char myhostname[256];
66         int r = gethostname(myhostname, sizeof(myhostname)-1);
67         if(r == -1) {
68             printf("Error getting hostname!\n");
69             QTimer::singleShot(0, this, SIGNAL(quit()));
70             return;
71         }
72         host = myhostname;
73         printf("Listening on %s:%d ...\n", host.toLatin1().data(), port);
74     }
75
76 private slots:
77     void serv_newConnection()
78     {
79         // Note: only 1 connection supported at a time in this example!
80         if(sock) {
81             delete nextPendingConnection();
82             printf("Connection ignored, already have one active.\n");
83             return;
84         }
85
86         printf("Connection received! Starting SASL handshake...\n");
87
88         sock = nextPendingConnection();
89         connect(sock, SIGNAL(disconnected()), SLOT(sock_connectionClosed()));
90         connect(sock, SIGNAL(readyRead()), SLOT(sock_readyRead()));
91         connect(sock, SIGNAL(error(QAbstractSocket::SocketError)), SLOT(sock_error(QAbstractSocket::SocketError x)));
92         connect(sock, SIGNAL(bytesWritten(qint64)), SLOT(sock_bytesWritten(qint64)));
93
94         sasl = new QCA::SASL;
95         connect(sasl, SIGNAL(authCheck(const QString &, const QString &)), SLOT(sasl_authCheck(const QString &, const QString &)));
96         connect(sasl, SIGNAL(nextStep(const QByteArray &)), SLOT(sasl_nextStep(const QByteArray &)));
97         connect(sasl, SIGNAL(authenticated()), SLOT(sasl_authenticated()));
98         connect(sasl, SIGNAL(readyRead()), SLOT(sasl_readyRead()));
99         connect(sasl, SIGNAL(readyReadOutgoing()), SLOT(sasl_readyReadOutgoing()));
100        connect(sasl, SIGNAL(error()), SLOT(sasl_error()));
101        connect(sasl, SIGNAL(serverStarted()), SLOT(sasl_serverStarted()));
102
103        mode = 0;
104        inbuf.resize(0);
105
106        sasl->setConstraints((QCA::SASL::AuthFlags) (QCA::SASL::AllowPlain | QCA::SASL::AllowAnonymous));
107
108        sasl->startServer(PROTO_NAME, host, realm);
109    }
110
111 signals:
112     void quit();
113
114 private slots:
115     void sasl_serverStarted()
116     {
117         sendLine(sasl->mechanismList().join(" "));
118     }
119
120     void sock_connectionClosed()
121     {
122         printf("Connection closed by peer.\n");
123         close();
124     }
125
126     void sock_error(QAbstractSocket::SocketError x)
127     {
128         printSocketError(x);
129         close();

```

```

130     }
131
132     void sock_readyRead()
133     {
134         if(sock->canReadLine()) {
135             QString line = sock->readLine();
136             line.truncate(line.length()-1); // chop the newline
137             handleLine(line);
138         }
139     }
140
141     void sock_bytesWritten(qint64 x)
142     {
143         if(mode == 2) {
144             toWrite -= x;
145             if(toWrite <= 0) {
146                 printf("Sent, closing.\n");
147                 close();
148             }
149         }
150     }
151
152     void sasl_nextStep(const QByteArray &stepData)
153     {
154         QString line = "C";
155         if(!stepData.isEmpty()) {
156             line += ',';
157             line += arrayToString(stepData.data());
158         }
159         sendLine(line);
160     }
161
162     void sasl_authCheck(const QString &user, const QString &authzid)
163     {
164         printf("AuthCheck: User: [%s], Authzid: [%s]\n", user.toLatin1().data(), authzid.toLatin1().data());
165         sasl->continueAfterAuthCheck();
166     }
167
168     void sasl_authenticated()
169     {
170         sendLine("A");
171         printf("Authentication success.\n");
172         ++mode;
173         printf("SSF: %d\n", sasl->ssf());
174         sendLine(str);
175     }
176
177     void sasl_readyRead()
178     {
179         QByteArray a = sasl->read();
180         int oldsize = inbuf.size();
181         inbuf.resize(oldsize + a.size());
182         memcpy(inbuf.data() + oldsize, a.data(), a.size());
183         processInbuf();
184     }
185
186     void sasl_readyReadOutgoing()
187     {
188         QByteArray a = sasl->readOutgoing();
189         toWrite = a.size();
190         sock->write(a.data(), a.size());
191     }
192
193     void sasl_error()
194     {
195         QCA::SASL::Error x = sasl->errorCode();
196         if(x == QCA::SASL::ErrorInit) {

```

```
197         printf("Problem starting up SASL\n");
198         quit();
199     }
200     else if(x == QCA::SASL::ErrorHandshake) {
201         sendLine("E");
202         printf("Authentication failed. AuthCondition = %d.\n", sasl->authCondition());
203         if ( sasl->authCondition() == QCA::SASL::NoUser ) {
204             printf( "No user!\n" );
205         }
206         close();
207     }
208     else if(x == QCA::SASL::ErrorCrypt) {
209         printf("SASL security layer error!\n");
210         close();
211     }
212 }
213
214
215 private:
216     QString host, realm;
217     int port;
218     QString str;
219     QByteArray inbuf;
220     int toWrite;
221     QTcpSocket *sock;
222     QCA::SASL *sasl;
223     int mode;
224
225
226     void processInbuf()
227     {
228     }
229
230     void handleLine(const QString &line)
231     {
232         printf("Reading: [%s]\n", line.toLatin1().data());
233         if(mode == 0) {
234             int n = line.indexOf(' ');
235             if(n != -1) {
236                 QString mech = line.mid(0, n);
237                 QString rest = line.mid(n+1).toUtf8();
238                 sasl->putServerFirstStep(mech, stringToArray(rest));
239             }
240             else
241                 sasl->putServerFirstStep(line);
242             ++mode;
243         }
244         else if(mode == 1) {
245             QString type, rest;
246             int n = line.indexOf(',');
247             if(n != -1) {
248                 type = line.mid(0, n);
249                 rest = line.mid(n+1);
250             }
251             else {
252                 type = line;
253                 rest = "";
254             }
255
256             if(type == "C") {
257                 sasl->putStep(stringToArray(rest));
258             }
259             else {
260                 printf("Bad format from peer, closing.\n");
261                 close();
262                 return;
263             }
264         }
265     }
```



```

264         }
265     }
266
267     void close()
268     {
269         delete sasl;
270         sock->deleteLater();
271         sock = 0;
272         sasl = 0;
273     }
274     QString arrayToString(const QByteArray &ba)
275     {
276         QCA::Base64 encoder;
277         return encoder.arrayToString(ba);
278     }
279
280     QByteArray stringToArray(const QString &s)
281     {
282         QCA::Base64 decoder(QCA::Decode);
283         return decoder.stringToArray(s).toByteArray();
284     }
285
286     void sendLine(const QString &line)
287     {
288         printf("Writing: {%s}\n", line.toUtf8().data());
289         QString s = line + '\n';
290         QByteArray a = s.toUtf8();
291         if(mode == 2)
292             sasl->write(a);
293         else
294             sock->write(a.data(), a.length());
295     }
296
297     void printSocketError(QAbstractSocket::SocketError x)
298     {
299         QString s;
300         if(x == QAbstractSocket::ConnectionRefusedError)
301             s = "connection refused or timed out";
302         else if(x == QAbstractSocket::RemoteHostClosedError)
303             s = "remote host closed the connection";
304         else if(x == QAbstractSocket::HostNotFoundError)
305             s = "host not found";
306         else if(x == QAbstractSocket::SocketAccessError)
307             s = "access error";
308         else if(x == QAbstractSocket::SocketResourceError)
309             s = "too many sockets";
310         else if(x == QAbstractSocket::SocketTimeoutError)
311             s = "operation timed out";
312         else if(x == QAbstractSocket::DatagramTooLargeError)
313             s = "datagram was larger than system limit";
314         else if(x == QAbstractSocket::NetworkError)
315             s = "network error";
316         else if(x == QAbstractSocket::AddressInUseError)
317             s = "address is already in use";
318         else if(x == QAbstractSocket::SocketAddressNotAvailableError)
319             s = "address does not belong to the host";
320         else if(x == QAbstractSocket::UnsupportedSocketOperationError)
321             s = "operation is not supported by the local operating system";
322         else
323             s = "unknown socket error";
324
325         printf("Socket error: %s\n", s.toLatin1().data());
326     }
327 };
328
329 #include "saslservtest.moc"
330

```

```
331 int main(int argc, char **argv)
332 {
333     QCA::Initializer init;
334     QApplication app(argc, argv);
335
336     QString host, user, pass;
337     QString str = "Hello, World";
338
339     if(argc >= 2)
340         str = argv[2];
341
342     if(!QCA::isSupported("sasl")) {
343         printf("SASL not supported!\n");
344         return 1;
345     }
346
347     QCA::setAppName("saslservtest");
348
349     ServerTest *s = new ServerTest(str, PROTO_PORT);
350     QObject::connect(s, SIGNAL(quit()), &app, SLOT(quit()));
351     s->start();
352     app.exec();
353     delete s;
354
355     return 0;
356 }
```

12.18 sasltest.cpp

The code below shows how to create a SASL client.

```

1  /*
2  Copyright (C) 2003-2006 Justin Karneges <justin@affinix.com>, Michail Pishchagin
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20  */
21
22  #include <QCoreApplication>
23  #include <QTimer>
24  #include <QTcpSocket>
25  #include <QTcpServer>
26  #include <stdio.h>
27
28  #ifdef Q_OS_UNIX
29  #include <unistd.h>
30  #endif
31
32  // QtCrypto has the declarations for all of QCA
33  #include <QtCrypto>
34
35  #define PROTO_NAME "foo"
36  #define PROTO_PORT 8001
37
38  static QString prompt(const QString &s)
39  {
40      printf("* %s ", s.toLatin1().data());
41      fflush(stdout);
42      char line[256];
43      fgets(line, 255, stdin);
44      QString result = line;
45      if(result[result.length()-1] == '\n')
46          result.truncate(result.length()-1);
47      return result;
48  }
49
50  class ClientTest : public QObject
51  {
52      Q_OBJECT
53  public:
54      ClientTest()
55      {
56          sock = new QTcpSocket;
57          connect(sock, SIGNAL(connected()), SLOT(sock_connected()));
58          connect(sock, SIGNAL(disconnected()), SLOT(sock_connectionClosed()));
59          connect(sock, SIGNAL(readyRead()), SLOT(sock_readyRead()));
60          connect(sock, SIGNAL(error(QAbstractSocket::SocketError)), SLOT(sock_error(QAbstractSocket::SocketError)));
61
62          sasl = new QCA::SASL;

```

```

63     connect(sasl, SIGNAL(clientStarted(bool, const QByteArray &)), SLOT(sasl_clientFirstStep(bool,
64     connect(sasl, SIGNAL(nextStep(const QByteArray &)), SLOT(sasl_nextStep(const QByteArray &)));
65     connect(sasl, SIGNAL(needParams(const QCA::SASL::Params &)), SLOT(sasl_needParams(const QCA::SA
66     connect(sasl, SIGNAL(authenticated()), SLOT(sasl_authenticated()));
67     connect(sasl, SIGNAL(readyRead()), SLOT(sasl_readyRead()));
68     connect(sasl, SIGNAL(readyReadOutgoing()), SLOT(sasl_readyReadOutgoing()));
69     connect(sasl, SIGNAL(error()), SLOT(sasl_error()));
70 }
71
72 void start(const QString &_host, int port, const QString &user="", const QString &pass="")
73 {
74     mode = 0;
75     host = _host;
76     sock->connectToHost(host, port);
77     sasl->setConstraints((QCA::SASL::AuthFlags)(QCA::SASL::AllowPlain | QCA::SASL::AllowAnonymous),
78
79     if(!user.isEmpty()) {
80         sasl->setUsername(user);
81         sasl->setAuthzid(user);
82     }
83     if(!pass.isEmpty())
84         sasl->setPassword(pass.toUtf8());
85 }
86
87 ~ClientTest()
88 {
89     delete sock;
90     delete sasl;
91 }
92
93 signals:
94     void quit();
95
96 private slots:
97     void sock_connected()
98     {
99         printf("Connected to server. Awaiting mechanism list...\n");
100     }
101
102     void sock_connectionClosed()
103     {
104         printf("Connection closed by peer.\n");
105         quit();
106     }
107
108     void sock_error(QAbstractSocket::SocketError x)
109     {
110         printSocketError(x);
111         quit();
112     }
113
114     void sock_readyRead()
115     {
116         if(mode == 2) {
117             int avail = sock->bytesAvailable();
118             QByteArray a(avail, 0);
119             int n = sock->read(a.data(), a.size());
120             a.resize(n);
121             printf("Read %d bytes\n", a.size());
122             sasl->writeIncoming(a);
123         }
124         else {
125             if(sock->canReadLine()) {
126                 QString line = sock->readLine();
127                 line.truncate(line.length()-1); // chop the newline
128                 handleLine(line);
129             }

```

```

130     }
131 }
132
133 void sasl_clientFirstStep(bool clientInit, const QByteArray &clientInitData)
134 {
135     ++mode;
136     printf("Choosing mech: %s\n", sasl->mechanism().toLatin1().data());
137     QString line = sasl->mechanism();
138     if(clientInit) {
139         line += ' ';
140         line += arrayToString(clientInitData);
141     }
142     sendLine(line);
143 }
144
145 void sasl_nextStep(const QByteArray &stepData)
146 {
147     QString line = "C";
148     if(!stepData.isEmpty()) {
149         line += ',';
150         line += arrayToString(stepData);
151     }
152     sendLine(line);
153 }
154
155 void sasl_needParams(const QCA::SASL::Params &params)
156 {
157     if(params.needUsername())
158         sasl->setUsername(prompt("Username:"));
159     if(params.canSendAuthzid()) {
160         QString authzid = prompt("Authorize As (enter to skip):");
161         if(!authzid.isEmpty())
162             sasl->setAuthzid(authzid);
163     }
164     if(params.needPassword()) {
165         QCA::ConsolePrompt prompt;
166         prompt.getHidden("* Password");
167         prompt.waitForFinished();
168         QCA::SecureArray pass = prompt.result();
169         sasl->setPassword(pass);
170     }
171     if(params.canSendRealm()) {
172         QStringList realms = sasl->realmList();
173         printf("Available realms:\n");
174         foreach(const QString &s, realms)
175             printf("  %s\n", qPrintable(s));
176         sasl->setRealm(prompt("Realm:"));
177     }
178     sasl->continueAfterParams();
179 }
180
181 void sasl_authenticated()
182 {
183     printf("SASL success!\n");
184     printf("SSF: %d\n", sasl->ssf());
185 }
186
187 void sasl_readyRead()
188 {
189     QByteArray a = sasl->read();
190     int oldsize = inbuf.size();
191     inbuf.resize(oldsize + a.size());
192     memcpy(inbuf.data() + oldsize, a.data(), a.size());
193     processInbuf();
194 }
195
196 void sasl_readyReadOutgoing()

```

```
197     {
198         QByteArray a = sasl->readOutgoing();
199         sock->write(a.data(), a.size());
200     }
201
202     void sasl_error()
203     {
204         printf("SASL error! Auth Condition = %d.\n", sasl->authCondition());
205         quit();
206         return;
207     }
208
209 private:
210     QTcpSocket *sock;
211     QCA::SASL *sasl;
212     int mode;
213     QString host;
214     QByteArray inbuf;
215
216     QString arrayToString(const QByteArray &ba)
217     {
218         QCA::Base64 encoder;
219         return encoder.arrayToString(ba);
220     }
221
222     QByteArray stringToArray(const QString &s)
223     {
224         QCA::Base64 decoder(QCA::Decode);
225         return decoder.stringToArray(s).toByteArray();
226     }
227
228     void sendLine(const QString &line)
229     {
230         printf("Writing: {%s}\n", line.toUtf8().data());
231         QString s = line + '\n';
232         QByteArray a = s.toUtf8();
233         if(mode == 2)
234             sasl->write(a);
235         else
236             sock->write(a.data(), a.length());
237     }
238
239     void printSocketError(QAbstractSocket::SocketError x)
240     {
241         QString s;
242         if(x == QAbstractSocket::ConnectionRefusedError)
243             s = "connection refused or timed out";
244         else if(x == QAbstractSocket::RemoteHostClosedError)
245             s = "remote host closed the connection";
246         else if(x == QAbstractSocket::HostNotFoundError)
247             s = "host not found";
248         else if(x == QAbstractSocket::SocketAccessError)
249             s = "access error";
250         else if(x == QAbstractSocket::SocketResourceError)
251             s = "too many sockets";
252         else if(x == QAbstractSocket::SocketTimeoutError)
253             s = "operation timed out";
254         else if(x == QAbstractSocket::DatagramTooLargeError)
255             s = "datagram was larger than system limit";
256         else if(x == QAbstractSocket::NetworkError)
257             s = "network error";
258         else if(x == QAbstractSocket::AddressInUseError)
259             s = "address is already in use";
260         else if(x == QAbstractSocket::SocketAddressNotAvailableError)
261             s = "address does not belong to the host";
262         else if(x == QAbstractSocket::UnsupportedSocketOperationError)
263             s = "operation is not supported by the local operating system";
```

```

264         else
265             s = "unknown socket error";
266         printf("Socket error: %s\n", s.toLatin1().data());
267     }
268
269     void processInbuf()
270     {
271         QStringList list;
272         for(int n = 0; n < (int)inbuf.size(); ++n) {
273             if(inbuf[n] == '\n') {
274                 list += QString::fromUtf8(inbuf.data(), n);
275
276                 char *p = inbuf.data();
277                 ++n;
278                 int x = inbuf.size() - n;
279                 memmove(p, p + n, x);
280                 inbuf.resize(x);
281
282                 // start over, basically
283                 n = -1;
284             }
285         }
286
287         foreach(QString line, list)
288             handleLine(line);
289     }
290
291     void handleLine(const QString &line)
292     {
293         printf("Reading: [%s]\n", line.toLatin1().data());
294         if(mode == 0) {
295             // first line is the method list
296             QStringList mechlist = line.split(' ');
297             sasl->startClient(PROTO_NAME, host, mechlist);
298         }
299         else if(mode == 1) {
300             QString type, rest;
301             int n = line.indexOf(',');
302             if(n != -1) {
303                 type = line.mid(0, n);
304                 rest = line.mid(n+1);
305             }
306             else {
307                 type = line;
308                 rest = "";
309             }
310
311             if(type == "C") {
312                 sasl->putStep(stringToArray(rest));
313             }
314             else if(type == "E") {
315                 printf("Authentication failed.\n");
316                 quit();
317                 return;
318             }
319             else if(type == "A") {
320                 printf("Authentication success.\n");
321                 ++mode;
322                 sock_readyRead(); // any extra data?
323                 return;
324             }
325             else {
326                 printf("Bad format from peer, closing.\n");
327                 quit();
328                 return;
329             }
330         }
331     }

```

```
331     }
332 };
333
334 #include "saslttest.moc"
335
336 void usage()
337 {
338     printf("usage: saslttest host [user] [pass]\n");
339 }
340
341 int main(int argc, char **argv)
342 {
343     QCA::Initializer init;
344     QApplication app(argc, argv);
345
346     QString host, user, pass;
347     QString str = "Hello, World";
348     if(argc < 2) {
349         usage();
350         return 0;
351     }
352     host = argv[1];
353     if(argc >= 3)
354         user = argv[2];
355     if(argc >= 4)
356         pass = argv[3];
357
358     if(!QCA::isSupported("sasl")) {
359         printf("SASL not supported!\n");
360         return 1;
361     }
362
363     ClientTest *c = new ClientTest;
364     QObject::connect(c, SIGNAL(quit()), &app, SLOT(quit()));
365     c->start(host, PROTO_PORT, user, pass);
366     app.exec();
367     delete c;
368
369     return 0;
370 }
```


12.19 sslservtest.cpp

The code below shows how to create an SSL server.

Note that this server returns a self-signed certificate for "example.com", and that the certificate is expired.

The design used here only allows for one connection at a time. If you want to allow for more, you should probably create a "TlsConnection" object that aggregates a [QCA::TLS](#) object and a [QTcpSocket](#) (plus a little bit of state information) that handles a single connection. Then just create a TlsConnection for each server connection.

```

1 /*
2  Copyright (C) 2003 Justin Karneges <justin@affinix.com>
3  Copyright (C) 2006 Brad Hards <bradh@frogmouth.net>
4
5  Permission is hereby granted, free of charge, to any person obtaining a copy
6  of this software and associated documentation files (the "Software"), to deal
7  in the Software without restriction, including without limitation the rights
8  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9  copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in
13 all copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
19 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
20 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
21 */
22
23 #include <QtCrypto>
24
25 #include <QCoreApplication>
26 #include <QDebug>
27 #include <QHostAddress>
28 #include <QTcpServer>
29 #include <QTcpSocket>
30 #include <QTimer>
31
32 char pemdata_cert[] =
33     "-----BEGIN CERTIFICATE-----\n"
34     "MIICeTCCAeKgAwIBAgIRAKKKnOj6Aarmwf0phApitVAwDQYJKoZIhvcNAQEFBQAw\n"
35     "ODELMakGA1UEBhMCMVVMxMDU1Ml0XDTA3MDMxNTA3MDU1Ml0wOjEVMjE0MjE0\n"
36     "eGFTcGx1IENBMjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0MjE0\n"
37     "A1UEAxMCMRhhbXBsZSBvc2VyMQswCQYDVQGEwJVUzEUMBIGA1UEChMLRhhbXBsZ\n"
38     "ZSBPcmcwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAPkKn0FfHMvRZv+3uFw\n"
39     "VrOadJmANzLVeVW/DH2p4CXokXSksM66ZMqFuQRBk5rnIZZpZmVp1tTRDvt9sEAY\n"
40     "YNa8CRM4HXkVlU0lCKdey18CSq2VuSvNtw8dDpoBmQt3nr9tePvKHnpS3nm6YjR2\n"
41     "NEvIKt1P4mHzYXLmwoF24C1bAgMBAAAGjgYAwfjAdBgNVHQ4EFgQUmQIdzyDaPYWF\n"
42     "fPJ8PPOmleSsucwHwYDVR0jBBGwFoAUkCglAizTO7iqwLeaO6r/8kJuqhMwDAYD\n"
43     "VR0TAQH/BAIwADAeBgNVHREEFzAVgRNleGFTcGx1QGV4YW1wbGUuY29tMA4GA1Ud\n"
44     "DwEB/wQEAwIF4DANBgkqhkiG9w0BAQUFAAOBgQAuhbiUgy2a++EUccaonID7eTJZ\n"
45     "F3D5qXMqUpQx1YxU8du+9AxDD7nFXTmkQC2pzfmEclznRNmJlZeLRL72VysVndcT\n"
46     "psyM8ABkvPpld2jWlyccVjGpt+/RN5IPKm/YIbtIZcywvWuXrOp1lanVmpplLfPnO\n"
47     "6yneBkC9iqjOv/+Q+A==\n"
48     "-----END CERTIFICATE-----\n";
49
50 char pemdata_privkey[] =
51     "-----BEGIN PRIVATE KEY-----\n"
52     "MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwgGJdAgEAAoGBAPkKn0FfHMvRZv+3\n"
53     "uFwVrOadJmANzLVeVW/DH2p4CXokXSksM66ZMqFuQRBk5rnIZZpZmVp1tTRDvt9\n"
54     "sEAYYNa8CRM4HXkVlU0lCKdey18CSq2VuSvNtw8dDpoBmQt3nr9tePvKHnpS3nm6\n"
55     "YjR2NEvIKt1P4mHzYXLmwoF24C1bAgMBAAECgYEAyIjJHDaeVXDU42zovyxpxE4n\n"

```

```

56     "PcOEryY+gdFJE8DFgUD4flhuFs4iCuNg+PaG42p+hf9IARNvSho/RcEaVg4AJrV\n"
57     "jRP8r7fSqcIGr6lGuvDFv3SU5ddy84g5oqLYGKvuPSHMGfVsZSxAWOrzD4bH19L\n"
58     "SNqtNcpdBsBd7ZiEE4ECQQD/oJGui9D5Dx3QVcS+QV4F8wuyN9jYIANmX/17o0fl\n"
59     "BL0bwRU4RICwadrcybi5N0JQLIYSUm2HGqNvAJbtuQxAKEA+WeYLLYPeawcy+WU\n"
60     "kGcOR7BUjHiG71+6cvU4XIDW2bezA04fqWXkZRFaWHTMpQb785/XalFftgS2lkql\n"
61     "8yLDSwJAHkeT2hwftdDPlEUEmBDAJW5DvWmWGu3u2G1cfbGZl9oUyhM7ixXHg57\n"
62     "6VlPs0jTZxHPE86FwNIr99MXDbCbKQJBAMDFQJK+ecGirXNP1P+0GA6DFSap9inJ\n"
63     "BRTbwx+EmgwX966DUOefEOSpbDIVVSPs/Qr2LgtIMEFA7Y0+j3wZD3cCQBsTwccd\n"
64     "ASQx59xakpq1leOlTYz14rjwodr4QMyj26WxEPJtz7hKokx/+EH6fWuPIUSrROM5\n"
65     "07y2gaVbYxtis0s=\n"
66     "-----END PRIVATE KEY-----\n";
67
68 class SecureServer : public QObject
69 {
70     Q_OBJECT
71
72 public:
73     enum { Idle, Handshaking, Active, Closing };
74
75     SecureServer(quint16 _port) : port(_port)
76     {
77         server = new QTcpServer;
78         connect( server, SIGNAL(newConnection()), SLOT(server_handleConnection()) );
79
80         ssl = new QCA::TLS;
81         connect(ssl, SIGNAL(handshaken()), SLOT(ssl_handshaken()));
82         connect(ssl, SIGNAL(readyRead()), SLOT(ssl_readyRead()));
83         connect(ssl, SIGNAL(readyReadOutgoing()), SLOT(ssl_readyReadOutgoing()));
84         connect(ssl, SIGNAL(closed()), SLOT(ssl_closed()));
85         connect(ssl, SIGNAL(error()), SLOT(ssl_error()));
86
87         cert = QCA::Certificate::fromPEM(pemdata_cert);
88         privkey = QCA::PrivateKey::fromPEM(pemdata_privkey);
89
90         mode = Idle;
91     }
92
93     ~SecureServer()
94     {
95         delete ssl;
96         delete server;
97     }
98
99     void start()
100     {
101         if(cert.isNull()) {
102             qDebug() << "Error loading cert!";
103             QTimer::singleShot(0, this, SIGNAL(quit()));
104             return;
105         }
106         if(privkey.isNull()) {
107             qDebug() << "Error loading private key!";
108             QTimer::singleShot(0, this, SIGNAL(quit()));
109             return;
110         }
111         if(false == server->listen(QHostAddress::Any, port)) {
112             qDebug() << "Error binding to port " << port;
113             QTimer::singleShot(0, this, SIGNAL(quit()));
114             return;
115         }
116         qDebug() << "Listening on port" << port;
117     }
118
119 signals:
120     void quit();
121
122 private slots:

```

```

123     void sock_readyRead()
124     {
125         QByteArray buf(sock->bytesAvailable(), 0x00);
126
127         int num = sock->read(buf.data(), buf.size());
128
129         if ( -1 == num )
130             qDebug() << "Error reading data from socket";
131
132         if (num < buf.size() )
133             buf.resize(num);
134
135         ssl->writeIncoming(buf);
136     }
137
138     void server_handleConnection()
139     {
140         // Note: only 1 connection supported at a time in this example!
141         if(mode != Idle) {
142             QTcpSocket* tmp = server->nextPendingConnection();
143             tmp->close();
144             connect(tmp, SIGNAL(disconnected()), tmp, SLOT(deleteLater()));
145             qDebug() << "throwing away extra connection";
146             return;
147         }
148         mode = Handshaking;
149         sock = server->nextPendingConnection();
150         connect(sock, SIGNAL(readyRead()), SLOT(sock_readyRead()));
151         connect(sock, SIGNAL(disconnected()), SLOT(sock_disconnected()));
152         connect(sock, SIGNAL(error(QAbstractSocket::SocketError)),
153             SLOT(sock_error(QAbstractSocket::SocketError)));
154         connect(sock, SIGNAL(bytesWritten(qint64)), SLOT(sock_bytesWritten(qint64)));
155
156         qDebug() << "Connection received! Starting TLS handshake.";
157         ssl->setCertificate(cert, privkey);
158         ssl->startServer();
159     }
160
161     void sock_disconnected()
162     {
163         qDebug() << "Connection closed.";
164     }
165
166     void sock_bytesWritten(qint64 x)
167     {
168         if(mode == Active && sent) {
169             qint64 bytes = ssl->convertBytesWritten(x);
170             bytesLeft -= bytes;
171
172             if(bytesLeft == 0) {
173                 mode = Closing;
174                 qDebug() << "Data transfer complete - SSL shutting down";
175                 ssl->close();
176             }
177         }
178     }
179
180     void sock_error(QAbstractSocket::SocketError error)
181     {
182         qDebug() << "Socket error: " << (unsigned) error;
183     }
184
185     void ssl_handshaken()
186     {
187         qDebug() << "Successful SSL handshake. Waiting for newline.";
188         bytesLeft = 0;
189         sent = false;

```

```
190         mode = Active;
191         ssl->continueAfterStep();
192     }
193
194     void ssl_readyRead()
195     {
196         QByteArray a = ssl->read();
197         QByteArray b =
198             "<html>\n"
199             "<head><title>Test</title></head>\n"
200             "<body>this is only a test</body>\n"
201             "</html>\n";
202
203         qDebug() << "Sending test response.";
204         sent = true;
205         ssl->write(b);
206     }
207
208     void ssl_readyReadOutgoing()
209     {
210         int plainBytes;
211         QByteArray outgoingData = ssl->readOutgoing(&plainBytes);
212         sock->write( outgoingData );
213     }
214
215     void ssl_closed()
216     {
217         qDebug() << "Closing socket.";
218         sock->close();
219         mode = Idle;
220     }
221
222     void ssl_error()
223     {
224         if(ssl->errorCode() == QCA::TLS::ErrorHandshake) {
225             qDebug() << "SSL Handshake Error! Closing.";
226             sock->close();
227         }
228         else {
229             qDebug() << "SSL Error! Closing.";
230             sock->close();
231         }
232         mode = Idle;
233     }
234
235 private:
236     quint16 port;
237     QTcpServer *server;
238     QTcpSocket *sock;
239     QCA::TLS *ssl;
240     QCA::Certificate cert;
241     QCA::PrivateKey privkey;
242
243     bool sent;
244     int mode;
245     qint64 bytesLeft;
246 };
247
248 #include "sslservtest.moc"
249
250 int main(int argc, char **argv)
251 {
252     QCA::Initializer init;
253
254     QCoreApplication app(argc, argv);
255     int port = argc > 1 ? QString(argv[1]).toInt() : 8000;
256
```

```
257     if(!QCA::isSupported("tls")) {
258         qDebug() << "TLS not supported!";
259         return 1;
260     }
261
262     SecureServer *server = new SecureServer(port);
263     QObject::connect(server, SIGNAL(quit()), &app, SLOT(quit()));
264     server->start();
265     app.exec();
266     delete server;
267
268     return 0;
269 }
```

12.20 ssltest.cpp

The code below shows how to create an SSL client

```

1  /*
2  Copyright (C) 2003-2005 Justin Karneges <justin@affinix.com>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20 */
21
22 #include <QtCrypto>
23
24 #include <QCoreApplication>
25 #include <QTcpSocket>
26
27 char exampleCA_cert[] =
28     "-----BEGIN CERTIFICATE-----\n"
29     "MIICSzCCAbSgAwIBAgIBADANBgkqhkiG9w0BAQUFADA4MRMwEQYDVQQDEwpFeGFt\n"
30     "cGx1IENBMQswCQYDVQQGEwJVUzEUMBIGA1UEChMLRXhhbXBsZSBPcmcwHhcNMDYw\n"
31     "MzE1MDY1ODMyWhcNMDYwNDE1MDY1ODMyWjA4MRMwEQYDVQQDEwpFeGFtY29w\n"
32     "MQswCQYDVQQGEwJVUzEUMBIGA1UEChMLRXhhbXBsZSBPcmcwZ8wDQYJKoZIhvcN\n"
33     "AQEBBQADgY0AMIGJAoGBAL6ULdOxmpeZ+G/ypV12eNO4qnHSVIPTrYPkQuweXqPy\n"
34     "atwGFheG+hLVsNIh9GGOS0tCe7a3hBBKN0BJglppfk2x39cDx7hefYqjBuZvp/00\n"
35     "8Ja3qlQiJLezITZKLxMBrsibcvcuH8zpfUdys2yaN+YGeqNfjQuoNN3By11TwuGJ\n"
36     "AgMBAAgJZTBjMB0GA1UdDgQWBBSQKCUCLNM7uKrAt5o7qv/yQm6qEzASBgNVHRMB\n"
37     "Af8ECDAGAQBAGeIMB4GA1UdEQQXMBWBE2V4Yw1wbGVAZXhhbXBsZS5jb20wDgYD\n"
38     "VR0PAQH/BAQDAgEGMA0GCSqGSIb3DQEBBQUAA4GBAAh+SIeT1Ao5qInw8oMS0Td0\n"
39     "lQ6h67ec/Jk5KmK4OoskuimmHI0Sp0C5kOCLehXbsVWW8pXsNC2fv0d2HkdaSUC\n"
40     "hwLzqgyZXd4mupIYlaOTZhuHDwWPCAOZS4LVsi2tndTRHKCP12441JjNKhmrZrhkR\n"
41     "u5zzd60nWgM9dKTaxuZM\n"
42     "-----END CERTIFICATE-----\n";
43
44 void showCertInfo(const QCA::Certificate &cert)
45 {
46     printf("-- Cert --\n");
47     printf("CN: %s\n", printable(cert.commonName()));
48     printf("Valid from: %s, until %s\n",
49           printable(cert.isValidBefore().toString()),
50           printable(cert.isValidAfter().toString()));
51     printf("PEM: %s\n", printable(cert.toPEM()));
52 }
53
54 static QString validityToString(QCA::Validity v)
55 {
56     QString s;
57     switch(v)
58     {
59         case QCA::ValidityGood:
60             s = "Validated";
61             break;
62         case QCA::ErrorRejected:

```

```

63         s = "Root CA is marked to reject the specified purpose";
64         break;
65     case QCA::ErrorUntrusted:
66         s = "Certificate not trusted for the required purpose";
67         break;
68     case QCA::ErrorSignatureFailed:
69         s = "Invalid signature";
70         break;
71     case QCA::ErrorInvalidCA:
72         s = "Invalid CA certificate";
73         break;
74     case QCA::ErrorInvalidPurpose:
75         s = "Invalid certificate purpose";
76         break;
77     case QCA::ErrorSelfSigned:
78         s = "Certificate is self-signed";
79         break;
80     case QCA::ErrorRevoked:
81         s = "Certificate has been revoked";
82         break;
83     case QCA::ErrorPathLengthExceeded:
84         s = "Maximum certificate chain length exceeded";
85         break;
86     case QCA::ErrorExpired:
87         s = "Certificate has expired";
88         break;
89     case QCA::ErrorExpiredCA:
90         s = "CA has expired";
91         break;
92     case QCA::ErrorValidityUnknown:
93     default:
94         s = "General certificate validation error";
95         break;
96     }
97     return s;
98 }
99
100 class SecureTest : public QObject
101 {
102     Q_OBJECT
103 public:
104     SecureTest()
105     {
106         sock_done = false;
107         ssl_done = false;
108
109         sock = new QTcpSocket;
110         connect(sock, SIGNAL(connected()), SLOT(sock_connected()));
111         connect(sock, SIGNAL(readyRead()), SLOT(sock_readyRead()));
112         connect(sock, SIGNAL(error(QAbstractSocket::SocketError)),
113                 SLOT(sock_error(QAbstractSocket::SocketError)));
114
115         ssl = new QCA::TLS;
116         connect(ssl, SIGNAL(certificateRequested()), SLOT(ssl_certificateRequested()));
117         connect(ssl, SIGNAL(handshaken()), SLOT(ssl_handshaken()));
118         connect(ssl, SIGNAL(readyRead()), SLOT(ssl_readyRead()));
119         connect(ssl, SIGNAL(readyReadOutgoing()),
120                 SLOT(ssl_readyReadOutgoing()));
121         connect(ssl, SIGNAL(closed()), SLOT(ssl_closed()));
122         connect(ssl, SIGNAL(error()), SLOT(ssl_error()));
123     }
124
125     ~SecureTest()
126     {
127         delete ssl;
128         delete sock;
129     }

```

```
130
131 void start(const QString &_host)
132 {
133     int n = _host.indexOf(':');
134     int port;
135     if(n != -1)
136     {
137         host = _host.mid(0, n);
138         port = _host.mid(n+1).toInt();
139     }
140     else
141     {
142         host = _host;
143         port = 443;
144     }
145
146     printf("Trying %s:%d...\n", qPrintable(host), port);
147     sock->connectToHost(host, port);
148 }
149
150 signals:
151     void quit();
152
153 private slots:
154     void sock_connected()
155     {
156         // We just do this to help doxygen...
157         QCA::TLS *ssl = SecureTest::ssl;
158
159         printf("Connected, starting TLS handshake...\n");
160
161         QCA::CertificateCollection rootCerts = QCA::systemStore();
162
163         // We add this one to show how, and to make it work with
164         // the server example.
165         rootCerts.addCertificate(QCA::Certificate::fromPEM(exampleCA_cert));
166
167         if(!QCA::haveSystemStore())
168             printf("Warning: no root certs\n");
169         else
170             ssl->setTrustedCertificates(rootCerts);
171
172         ssl->startClient(host);
173     }
174
175     void sock_readyRead()
176     {
177         // We just do this to help doxygen...
178         QCA::TLS *ssl = SecureTest::ssl;
179
180         ssl->writeIncoming(sock->readAll());
181     }
182
183     void sock_connectionClosed()
184     {
185         printf("\nConnection closed.\n");
186         sock_done = true;
187
188         if(ssl_done && sock_done)
189             emit quit();
190     }
191
192     void sock_error(QAbstractSocket::SocketError x)
193     {
194         if(x == QAbstractSocket::RemoteHostClosedError)
195         {
196             sock_connectionClosed();
```



```

197         return;
198     }
199
200     printf("\nSocket error.\n");
201     emit quit();
202 }
203
204 void ssl_handshaken()
205 {
206     // We just do this to help doxygen...
207     QCA::TLS *ssl = SecureTest::ssl;
208
209     QCA::TLS::IdentityResult r = ssl->peerIdentityResult();
210
211     printf("Successful SSL handshake using %s (%i of %i bits)\n",
212           qPrintable(ssl->cipherSuite()),
213           ssl->cipherBits(),
214           ssl->cipherMaxBits() );
215     if(r != QCA::TLS::NoCertificate)
216     {
217         cert = ssl->peerCertificateChain().primary();
218         if(!cert.isNull())
219             showCertInfo(cert);
220     }
221
222     QString str = "Peer Identity: ";
223     if(r == QCA::TLS::Valid)
224         str += "Valid";
225     else if(r == QCA::TLS::HostMismatch)
226         str += "Error: Wrong certificate";
227     else if(r == QCA::TLS::InvalidCertificate)
228         str += "Error: Invalid certificate.\n -> Reason: " +
229             validityToString(ssl->peerCertificateValidity());
230     else
231         str += "Error: No certificate";
232     printf("%s\n", qPrintable(str));
233
234     ssl->continueAfterStep();
235
236     printf("Let's try a GET request now.\n");
237     QString req = "GET / HTTP/1.0\nHost: " + host + "\n\n";
238     ssl->write(req.toLatin1());
239 }
240
241 void ssl_certificateRequested()
242 {
243     // We just do this to help doxygen...
244     QCA::TLS *ssl = SecureTest::ssl;
245
246     printf("Server requested client certificate.\n");
247     QList<QCA::CertificateInfoOrdered> issuerList = ssl->issuerList();
248     if(!issuerList.isEmpty())
249     {
250         printf("Allowed issuers:\n");
251         foreach(QCA::CertificateInfoOrdered i, issuerList)
252             printf("  %s\n", qPrintable(i.toString()));
253     }
254
255     ssl->continueAfterStep();
256 }
257
258 void ssl_readyRead()
259 {
260     // We just do this to help doxygen...
261     QCA::TLS *ssl = SecureTest::ssl;
262
263     QByteArray a = ssl->read();

```

```

264         printf("%s", a.data());
265     }
266
267     void ssl_readyReadOutgoing()
268     {
269         // We just do this to help doxygen...
270         QCA::TLS *ssl = SecureTest::ssl;
271
272         sock->write(ssl->readOutgoing());
273     }
274
275     void ssl_closed()
276     {
277         printf("SSL session closed.\n");
278         ssl_done = true;
279
280         if(ssl_done && sock_done)
281             emit quit();
282     }
283
284     void ssl_error()
285     {
286         // We just do this to help doxygen...
287         QCA::TLS *ssl = SecureTest::ssl;
288
289         int x = ssl->errorCode();
290         if(x == QCA::TLS::ErrorHandshake)
291         {
292             printf("SSL Handshake Error!\n");
293             emit quit();
294         }
295         else
296         {
297             printf("SSL Error!\n");
298             emit quit();
299         }
300     }
301
302 private:
303     QString host;
304     QTcpSocket *sock;
305     QCA::TLS *ssl;
306     QCA::Certificate cert;
307     bool sock_done, ssl_done;
308 };
309
310 #include "ssltest.moc"
311
312 int main(int argc, char **argv)
313 {
314     QCA::Initializer init;
315
316     QCoreApplication app(argc, argv);
317     QString host = argc > 1 ? argv[1] : "andbit.net";
318
319     if(!QCA::isSupported("tls"))
320     {
321         printf("TLS not supported!\n");
322         return 1;
323     }
324
325     SecureTest *s = new SecureTest;
326     QObject::connect(s, SIGNAL(quit()), &app, SLOT(quit()));
327     s->start(host);
328     app.exec();
329     delete s;
330

```

```
331         return 0;  
332     }
```

12.21 tlssocket.cpp

The code below shows how to create a socket that can operate over an Transport Layer Security (TLS, also known as SSL) connection.

```

1  /*
2  Copyright (C) 2007 Justin Karneges <justin@affinix.com>
3
4  Permission is hereby granted, free of charge, to any person obtaining a copy
5  of this software and associated documentation files (the "Software"), to deal
6  in the Software without restriction, including without limitation the rights
7  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  copies of the Software, and to permit persons to whom the Software is
9  furnished to do so, subject to the following conditions:
10
11  The above copyright notice and this permission notice shall be included in
12  all copies or substantial portions of the Software.
13
14  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17  AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
18  AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
19  CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20  */
21
22  #include "tlssocket.h"
23
24  class TLSSocket::Private : public QObject
25  {
26      Q_OBJECT
27  public:
28      TLSSocket *q;
29      QTcpSocket *sock;
30      QCA::TLS *tls;
31      QString host;
32      bool encrypted;
33      bool error, done;
34      QByteArray readbuf, writebuf;
35      QCA::Synchronizer sync;
36      bool waiting;
37
38      Private(TLSSocket *_q) : QObject(_q), q(_q), sync(_q)
39      {
40          sock = new QTcpSocket(this);
41          connect(sock, SIGNAL(connected()), SLOT(sock_connected()));
42          connect(sock, SIGNAL(readyRead()), SLOT(sock_readyRead()));
43          connect(sock, SIGNAL(bytesWritten(qint64)), SLOT(sock_bytesWritten(qint64)));
44          connect(sock, SIGNAL(error(QAbstractSocket::SocketError)), SLOT(sock_error(QAbstractSocket::SocketError)));
45
46          tls = new QCA::TLS(this);
47          connect(tls, SIGNAL(handshaken()), SLOT(tls_handshaken()));
48          connect(tls, SIGNAL(readyRead()), SLOT(tls_readyRead()));
49          connect(tls, SIGNAL(readyReadOutgoing()), SLOT(tls_readyReadOutgoing()));
50          connect(tls, SIGNAL(closed()), SLOT(tls_closed()));
51          connect(tls, SIGNAL(error()), SLOT(tls_error()));
52          tls->setTrustedCertificates(QCA::systemStore());
53          encrypted = false;
54          error = false;
55          waiting = false;
56          done = false;
57      }
58
59      bool waitForReadyRead(int msec)
60      {
61          waiting = true;

```

```

62         bool ok = sync.waitForCondition(msecs);
63         //while(1)
64         //    QCoreApplication::instance()->processEvents();
65         waiting = false;
66         if(error || done)
67             return false;
68         return ok;
69     }
70
71 private slots:
72     void sock_connected()
73     {
74         //printf("sock connected\n");
75         tls->startClient(host);
76     }
77
78     void sock_readyRead()
79     {
80         //printf("sock ready read\n");
81         QByteArray buf = sock->readAll();
82         //printf("%d bytes\n", buf.size());
83         tls->writeIncoming(buf);
84     }
85
86     void sock_bytesWritten(qint64 x)
87     {
88         Q_UNUSED(x);
89         //printf("sock bytes written: %d\n", (int)x);
90     }
91
92     void sock_error(QAbstractSocket::SocketError x)
93     {
94         //printf("sock error: %d\n", x);
95         Q_UNUSED(x);
96         done = true;
97         if(waiting)
98             sync.conditionMet();
99     }
100
101     void tls_handshaken()
102     {
103         //printf("tls handshaken\n");
104         if(tls->peerIdentityResult() != QCA::TLS::Valid)
105         {
106             printf("not valid\n");
107             sock->abort();
108             tls->reset();
109             error = true;
110         }
111         else
112         {
113             //printf("valid\n");
114             encrypted = true;
115             //printf("%d bytes in writebuf\n", writebuf.size());
116             if(!writebuf.isEmpty())
117             {
118                 //printf("[%s]\n", writebuf.data());
119                 tls->write(writebuf);
120                 writebuf.clear();
121             }
122         }
123         if(waiting)
124             sync.conditionMet();
125     }
126
127     void tls_readyRead()
128     {

```

```

129         //printf("tls ready read\n");
130         if(waiting)
131             sync.conditionMet();
132     }
133
134     void tls_readyReadOutgoing()
135     {
136         //printf("tls ready read outgoing\n");
137         QByteArray buf = tls->readOutgoing();
138         //printf("%d bytes\n", buf.size());
139         sock->write(buf);
140     }
141
142     void tls_closed()
143     {
144         //printf("tls closed\n");
145     }
146
147     void tls_error()
148     {
149         //printf("tls error\n");
150     }
151 };
152
153 TLSSocket::TLSSocket(QObject *parent)
154 :QTcpSocket(parent)
155 {
156     d = new Private(this);
157 }
158
159
160 TLSSocket::~TLSSocket()
161 {
162     delete d;
163 }
164
165 void TLSSocket::connectToHostEncrypted(const QString &host, quint16 port)
166 {
167     d->host = host;
168     setOpenMode(QIODevice::ReadWrite);
169     d->sock->connectToHost(host, port);
170 }
171
172 QCA::TLS *TLSSocket::tls()
173 {
174     return d->tls;
175 }
176
177 bool TLSSocket::waitForReadyRead(int msec)
178 {
179     /*if(d->readbuf.isEmpty())
180         return false;
181
182     if(d->tls->bytesAvailable() == 0)
183         return false;*/
184
185     return d->waitForReadyRead(msec);
186 }
187
188 quint64 TLSSocket::readData(char *data, quint64 maxlen)
189 {
190     if(!d->error)
191         d->readbuf += d->tls->read();
192     unsigned char *p = (unsigned char *)d->readbuf.data();
193     int size = d->readbuf.size();
194     int readsize = qMin(size, (int)maxlen);
195     int newsize = size - readsize;

```

```
196         memcpy(data, p, readsize);
197         memmove(p, p + readsize, newsize);
198         d->readbuf.resize(newsize);
199         return readsize;
200     }
201
202     qint64 TLSSocket::writeData(const char *data, qint64 len)
203     {
204         //printf("write %d bytes\n", (int)len);
205         QByteArray buf(data, len);
206         if(d->encrypted)
207             d->tls->write(buf);
208         else
209             d->writebuf += buf;
210         return len;
211     }
212
213     #include "tlsocket.moc"
```


Chapter 13

Qt Cryptographic Architecture Page Documentation

13.1 Architecture

Note:

You don't need to understand any of this to use QCA. It is documented for those who are curious, and for anyone planning to extend or modify QCA.

The design of QCA is based on the Bridge design pattern. The intent of the Bridge pattern is to "Decouple an abstraction from its implementation so that the two can vary independently." [Gamma et.al, pg 151].

To understand how this decoupling works in the case of QCA, it is easiest to look at an example - a cryptographic Hash. The API is pretty simple (although I've left out some parts that aren't required for this example):

```
class QCA_EXPORT Hash : public Algorithm, public BufferedComputation
{
public:
    Hash(const QString &type, const QString &provider);
    virtual void clear();
    virtual void update(const QCA::SecureArray &a);
    virtual QCA::SecureArray final();
}
```

The implementation for the Hash class is almost as simple:

```
Hash::Hash(const QString &type, const QString &provider)
:Algorithm(type, provider)
{
}

void Hash::clear()
{
    static_cast<HashContext *>(context())->clear();
}

void Hash::update(const QCA::SecureArray &a)
{
    static_cast<HashContext *>(context())->update(a);
}
```

```
QCA::SecureArray Hash::final()
{
    return static_cast<HashContext *>(context())->final();
}
```

The reason why it looks so simple is that the various methods in Hash just call out to equivalent routines in the context() object. The context comes from a call (getContext()) that is made as part of the Algorithm constructor. That getContext() call causes QCA to work through the list of providers (generally plugins) that it knows about, looking for a provider that can produce the right kind of context (in this case, a HashContext).

The code for a HashContext doesn't need to be linked into QCA - it can be varied in its implementation, including being changed at run-time. The application doesn't need to know how HashContext is implemented, because it just has to deal with the Hash class interface. In fact, HashContext may not be implemented, so the application should check (using [QCA::isSupported\(\)](#)) before trying to use features that are implemented with plugins.

The code for one implementation (in this case, calling OpenSSL) is shown below.

```
class opensslHashContext : public HashContext
{
public:
    opensslHashContext(const EVP_MD *algorithm, Provider *p, const QString &type) : HashContext(p, type)
    {
        m_algorithm = algorithm;
        EVP_DigestInit( &m_context, m_algorithm );
    };

    ~opensslHashContext()
    {
        EVP_MD_CTX_cleanup(&m_context);
    }

    void clear()
    {
        EVP_MD_CTX_cleanup(&m_context);
        EVP_DigestInit( &m_context, m_algorithm );
    }

    void update(const QCA::SecureArray &a)
    {
        EVP_DigestUpdate( &m_context, (unsigned char*)a.data(), a.size() );
    }

    QCA::SecureArray final()
    {
        QCA::SecureArray a( EVP_MD_size( m_algorithm ) );
        EVP_DigestFinal( &m_context, (unsigned char*)a.data(), 0 );
        return a;
    }

    Provider::Context *clone() const
    {
        return new opensslHashContext(*this);
    }

protected:
    const EVP_MD *m_algorithm;
    EVP_MD_CTX m_context;
};
```

This approach (using an Adapter pattern) is very common in QCA backends, because the plugins are often based on existing libraries.

In addition to the various Context objects, each provider also has a parameterised Factory class that has a `createContext()` method, as shown below:

```
Context *createContext(const QString &type)
{
    //OpenSSL_add_all_digests();
    if ( type == "sha1" )
        return new opensslHashContext( EVP_sha1(), this, type);
    else if ( type == "sha0" )
        return new opensslHashContext( EVP_sha(), this, type);
    else if ( type == "md5" )
        return new opensslHashContext( EVP_md5(), this, type);
    else if ( type == "aes128-cfb" )
        return new opensslCipherContext( EVP_aes_128_cfb(), 0, this, type);
    else if ( type == "aes128-cbc" )
        return new opensslCipherContext( EVP_aes_128_cbc(), 0, this, type);
    else
        return 0;
}
```

The resulting effect is that QCA can ask the provider to provide an appropriate Context object without worrying about how it is implemented.

13.2 Providers

QCA works on the concept of a "provider".

There is a limited internal provider (named "default"), but most of the work is done in plugin modules.

The logic to selection of a provider is fairly simple. The user can specify a provider name - if that name exists, and the provider supports the requested feature, then the named provider is used. If that didn't work, then the available plugins are searched (based on a priority order) for the requested feature. If that doesn't work, then the default provider is searched for the requested feature.

So the only way to get the default provider is to either have no other support whatsoever, or to specify the default provider directly (this goes for the algorithm constructors as well as `setGlobalRNG()`).

You can add your own provider in two ways - as a shared object plugin, and as a part of the client code.

The shared object plugin needs to be able to be found using the built-in scan logic - this normally means you need to install it into the `plugins/crypto` subdirectory within the directory that Qt is installed to. This will make it available for all applications.

If you have a limited application domain (such as a specialist algorithm, or a need to be bug-compatible), you may find it easier to create a client-side provider, and add it using the `QCA::insertProvider` call. There is an example of this - see [the AES-CMAC example](#).

13.3 Hashing Algorithms

There are a range of hashing algorithms available in QCA.

Hashing algorithms are used with the [Hash](#) and [MessageAuthenticationCode](#) classes.

The MD2 algorithm takes an arbitrary data stream, known as the message and outputs a condensed 128 bit (16 byte) representation of that data stream, known as the message digest. This algorithm is considered slightly more secure than MD5, but is more expensive to compute. Unless backward compatibility or interoperability are considerations, you are better off using the SHA1 or RIPEMD160 hashing algorithms. For more information on MD2, see B. Kalinski RFC1319 "The MD2 Message-Digest Algorithm". The label for MD2 is "md2".

The MD4 algorithm takes an arbitrary data stream, known as the message and outputs a condensed 128 bit (16 byte) representation of that data stream, known as the message digest. MD4 is not considered to be secure, based on known attacks. It should only be used for applications where collision attacks are not a consideration (for example, as used in the rsync algorithm for fingerprinting blocks of data). If a secure hash is required, you are better off using the SHA1 or RIPEMD160 hashing algorithms. MD2 and MD5 are both stronger 128 bit hashes. For more information on MD4, see R. Rivest RFC1320 "The %MD4 Message-Digest Algorithm". The label for MD4 is "md4".

The MD5 takes an arbitrary data stream, known as the message and outputs a condensed 128 bit (16 byte) representation of that data stream, known as the message digest. MD5 is not considered to be secure, based on known attacks. It should only be used for applications where collision attacks are not a consideration. If a secure hash is required, you are better off using the SHA1 or RIPEMD160 hashing algorithms. For more information on MD5, see R. Rivest RFC1321 "The MD5 Message-Digest Algorithm". The label for MD5 is "md5".

The RIPEMD160 algorithm takes an arbitrary data stream, known as the message (up to 2^{64} bits in length) and outputs a condensed 160 bit (20 byte) representation of that data stream, known as the message digest. The RIPEMD160 algorithm is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. The label for RIPEMD160 is "ripemd160".

The SHA-0 algorithm is a 160 bit hashing function, no longer recommended for new applications because of known (partial) attacks against it. The label for SHA-0 is "sha0".

The SHA-1 algorithm takes an arbitrary data stream, known as the message (up to 2^{64} bits in length) and outputs a condensed 160 bit (20 byte) representation of that data stream, known as the message digest. SHA-1 is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. For more information on the SHA-1 algorithm, see Federal Information Processing Standard Publication 180-2 "Specifications for the Secure Hash Standard", available from <http://csrc.nist.gov/publications/>. The label for SHA-1 is "sha1".

The SHA-224 algorithm takes an arbitrary data stream, known as the message (up to 2^{64} bits in length) and outputs a condensed 224 bit (28 byte) representation of that data stream, known as the message digest. SHA-224 is a "cut down" version of SHA-256, and you may be better off using SHA-256 in new designs. The SHA-224 algorithm is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. For more information on SHA-224, see Federal Information Processing Standard Publication 180-2 "Specifications for the Secure Hash Standard", with change notice 1, available from <http://csrc.nist.gov/publications/>. The label for SHA-224 is "sha224".

The SHA-256 algorithm takes an arbitrary data stream, known as the message (up to 2^{64} bits in length) and outputs a condensed 256 bit (32 byte) representation of that data stream, known as the message digest. The SHA-256 algorithm is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. For more information on SHA-256, see Federal Information Processing Standard Publication 180-2 "Specifications for the Secure Hash Standard", available from <http://csrc.nist.gov/publications/>. The label for SHA-256 is "sha256".

The SHA-384 algorithm takes an arbitrary data stream, known as the message (up to 2^{128} bits in length) and outputs a condensed 384 bit (48 byte) representation of that data stream, known as the message digest. The SHA-384 algorithm is a "cut down" version of SHA-512, and you may be better off using SHA-512 in new designs. The SHA-384 algorithm is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. For more information on SHA-384, see Federal Information Processing Standard Publication 180-2 "Specifications for the Secure %Hash Standard", available from <http://csrc.nist.gov/publications/>. The label for SHA-384 is "sha384".

The SHA-512 algorithm takes an arbitrary data stream, known as the message (up to 2^{128} bits in length) and outputs a condensed 512 bit (64 byte) representation of that data stream, known as the message digest. The SHA-512 algorithm is considered secure in that it is considered computationally infeasible to find the message that produced the message digest. For more information on SHA-512, see Federal Information Processing Standard Publication 180-2 "Specifications for the Secure %Hash Standard", available from <http://csrc.nist.gov/publications/>. The label for SHA-512 is "sha512".

Index

- ~Event
 - QCA::Event, [135](#)
- ~TLS
 - QCA::TLS, [296](#)
- _dir
 - QCA::TextFilter, [292](#)
- AACompromise
 - QCA::CRLEntry, [120](#)
- AbstractLogDevice
 - QCA::AbstractLogDevice, [48](#)
- accepted
 - QCA::PasswordAsker, [204](#)
 - QCA::TokenAsker, [308](#)
- addCertificate
 - QCA::CertificateCollection, [81](#)
- addCRL
 - QCA::CertificateCollection, [81](#)
- Alert
 - QCA::Logger, [191](#)
- Algorithm
 - QCA::Algorithm, [50](#)
- AlternativeName
 - QCA::CertificateInfoType, [88](#)
- append
 - QCA::CertificateCollection, [82](#)
 - QCA::SecureArray, [263](#)
- appendPluginDiagnosticText
 - QCA, [40](#)
- Application
 - QCA::KeyStore, [173](#)
- appName
 - QCA, [42](#)
- arrayToHex
 - QCA, [42](#)
- arrayToString
 - QCA::TextFilter, [291](#)
- Ascii
 - QCA::SecureMessage, [272](#)
- ask
 - QCA::PasswordAsker, [204](#)
 - QCA::TokenAsker, [308](#)
- at
 - QCA::MemoryRegion, [196](#), [197](#)
 - QCA::SecureArray, [261](#)
- authCheck
 - QCA::SASL, [255](#)
- AuthCondition
 - QCA::SASL, [248](#)
- authCondition
 - QCA::SASL, [252](#)
- authenticated
 - QCA::SASL, [255](#)
- AuthFail
 - QCA::SASL, [248](#)
- AuthFlags
 - QCA::SASL, [248](#)
- BadAuth
 - QCA::SASL, [248](#)
- BadProtocol
 - QCA::SASL, [248](#)
- BadServer
 - QCA::SASL, [248](#)
- Base64
 - QCA::Base64, [53](#)
- BasicContext
 - QCA::BasicContext, [55](#)
- BigInteger
 - QCA::BigInteger, [58](#)
- Binary
 - QCA::SecureMessage, [272](#)
- bitSize
 - QCA::PKey, [218](#)
- blockingEnabled
 - QCA::KeyGenerator, [164](#)
- blockSize
 - QCA::Cipher, [106](#)
- bundleSignerEnabled
 - QCA::SecureMessage, [273](#)
- busyFinished
 - QCA::KeyStoreManager, [189](#)
- bytesAvailable
 - QCA::SASL, [254](#)
 - QCA::SecureLayer, [266](#)
 - QCA::SecureMessage, [276](#)
 - QCA::TLS, [302](#)
- bytesOutgoingAvailable
 - QCA::SASL, [254](#)
 - QCA::SecureLayer, [266](#)

- QCA::TLS, 303
- bytesWritten
 - QCA::SecureMessage, 278
- CACompromise
 - QCA::CRLEntry, 120
- cancel
 - QCA::PasswordAsker, 204
 - QCA::TokenAsker, 308
- canClearsign
 - QCA::SecureMessage, 273
- canCompress
 - QCA::TLS, 299
- canDecrypt
 - QCA::PrivateKey, 224
- canEncrypt
 - QCA::PublicKey, 232
- canExport
 - QCA::PKey, 219
- canKeyAgree
 - QCA::PKey, 219
- canSendAuthzid
 - QCA::SASL::Params, 257
- canSendRealm
 - QCA::SASL::Params, 257
- canSetHostName
 - QCA::TLS, 299
- canSign
 - QCA::PrivateKey, 224
- canSignAndEncrypt
 - QCA::SecureMessage, 273
- canSignMultiple
 - QCA::SecureMessage, 272
- canUseFormat
 - QCA::CertificateRequest, 98
- canUsePKCS7
 - QCA::CertificateCollection, 82
- canVerify
 - QCA::PublicKey, 233
- CBC
 - QCA::Cipher, 104
- Certificate
 - QCA::Certificate, 66, 67
- certificate
 - QCA::CertificateAuthority, 75
 - QCA::KeyStoreEntry, 181
- CertificateAuthority
 - QCA::CertificateAuthority, 74, 75
- CertificateChain
 - QCA::CertificateChain, 78
- certificateChain
 - QCA::KeyBundle, 157
- CertificateCollection
 - QCA::CertificateCollection, 81
- CertificateHold
 - QCA::CRLEntry, 120
- CertificateInfo
 - QCA, 30
- CertificateInfoPair
 - QCA::CertificateInfoPair, 85
- CertificateInfoType
 - QCA::CertificateInfoType, 88
- CertificateInfoTypeKnown
 - QCA, 30
- CertificateOptions
 - QCA::CertificateOptions, 91
- CertificateRequest
 - QCA::CertificateRequest, 97
- certificateRequested
 - QCA::TLS, 305
- CertificateRequestFormat
 - QCA, 30
- certificates
 - QCA::CertificateCollection, 81
- CFB
 - QCA::Cipher, 104
- challenge
 - QCA::CertificateOptions, 92
 - QCA::CertificateRequest, 99
- change
 - QCA::Algorithm, 51
 - QCA::Certificate, 73
 - QCA::CertificateRequest, 102
 - QCA::CRL, 118
- Cipher
 - QCA::Cipher, 105
- cipherBits
 - QCA::TLS, 301
- cipherMaxBits
 - QCA::TLS, 301
- cipherSuite
 - QCA::TLS, 301
- clear
 - QCA::Base64, 53
 - QCA::BufferedComputation, 63
 - QCA::Cipher, 106
 - QCA::Filter, 143
 - QCA::Hash, 147
 - QCA::Hex, 151
 - QCA::MessageAuthenticationCode, 199
 - QCA::SecureArray, 260
- clearDiagnosticText
 - QCA::KeyStoreManager, 188
- clearPluginDiagnosticText
 - QCA, 40
- Clearsign
 - QCA::SecureMessage, 271
- ClientAuth

- QCA, 32
- ClientSendMode
 - QCA::SASL, 249
- close
 - QCA::SecureLayer, 266
 - QCA::TLS, 303
- closed
 - QCA::SecureLayer, 268
- CMS
 - QCA::CMS, 109
 - QCA::SecureMessage, 271
- CodeSigning
 - QCA, 32
- CommonName
 - QCA, 31
- commonName
 - QCA::Certificate, 69
- compare
 - QCA::BigInteger, 61
- complete
 - QCA::CertificateChain, 78
- compressionEnabled
 - QCA::TLS, 299
- constData
 - QCA::MemoryRegion, 196
 - QCA::SecureArray, 261
- Constraints
 - QCA, 30
- constraints
 - QCA::Certificate, 69
 - QCA::CertificateOptions, 92
 - QCA::CertificateRequest, 99
- ConstraintType
 - QCA::ConstraintType, 112
- ConstraintTypeKnown
 - QCA, 31
- context
 - QCA::Algorithm, 51
- continueAfterAuthCheck
 - QCA::SASL, 253
- continueAfterParams
 - QCA::SASL, 253
- continueAfterStep
 - QCA::TLS, 300
- convertBytesWritten
 - QCA::SASL, 255
 - QCA::SecureLayer, 267
 - QCA::TLS, 304
- ConvertGood
 - QCA, 34
- ConvertResult
 - QCA, 34
- convertResult
 - QCA::KeyLoader, 171
- Country
 - QCA, 31
- create
 - QCA::QPipe, 237
- createCertificate
 - QCA::CertificateAuthority, 75
- createCRL
 - QCA::CertificateAuthority, 75
- createDH
 - QCA::KeyGenerator, 165
- createDLGroup
 - QCA::KeyGenerator, 165
- createDSA
 - QCA::KeyGenerator, 165
- createRSA
 - QCA::KeyGenerator, 164
- creationDate
 - QCA::PGPKey, 212
- Critical
 - QCA::Logger, 191
- CRL
 - QCA::CRL, 115
- crl
 - QCA::KeyStoreEntry, 181
- CRLEntry
 - QCA::CRLEntry, 120
- crlLocations
 - QCA::Certificate, 69
 - QCA::CertificateOptions, 92
- crls
 - QCA::CertificateCollection, 81
- CRLSign
 - QCA, 31
- currentLogDevices
 - QCA::Logger, 192
- d
 - QCA::RSAPrivateKey, 243
- Data
 - QCA::Event, 134
- data
 - QCA::MemoryRegion, 196
 - QCA::SecureArray, 261
- DataEncipherment
 - QCA, 31
- Datagram
 - QCA::TLS, 295
- Debug
 - QCA::Logger, 191
- DecipherOnly
 - QCA, 31
- Decode
 - QCA, 33
- decode

- QCA::TextFilter, 291
- decodeString
 - QCA::TextFilter, 292
- decrypt
 - QCA::PrivateKey, 225
- defaultFeatures
 - QCA, 38
- DefaultFormat
 - QCA, 34
- DefaultPadding
 - QCA::Cipher, 104
- defaultProvider
 - QCA, 40
- deinit
 - QCA, 36
- deriveKey
 - QCA::PrivateKey, 226
- DERSequence
 - QCA, 34
- Detached
 - QCA::SecureMessage, 271
- DH
 - QCA::PKey, 216
- DHPrivateKey
 - QCA::DHPrivateKey, 123
- DHPublicKey
 - QCA::DHPublicKey, 125
- diagnosticText
 - QCA::KeyStoreManager, 188
 - QCA::SecureMessage, 278
- DigitalSignature
 - QCA, 31
- Direction
 - QCA, 33
- direction
 - QCA::Cipher, 105
 - QCA::TextFilter, 291
- Disabled
 - QCA::SASL, 248
- DLGroup
 - QCA::DLGroup, 126, 127
- dlGroup
 - QCA::KeyGenerator, 165
- DLGroupSet
 - QCA, 34
- DN
 - QCA::CertificateInfoType, 88
- dnOnly
 - QCA::CertificateInfoOrdered, 84
- DNS
 - QCA, 31
- domain
 - QCA::DHPrivateKey, 123
 - QCA::DHPublicKey, 125
 - QCA::DSAPrivateKey, 130
 - QCA::DSAPublicKey, 132
- DSA
 - QCA::PKey, 216
- DSA_1024
 - QCA, 35
- DSA_512
 - QCA, 35
- DSA_768
 - QCA, 35
- DSAPrivateKey
 - QCA::DSAPrivateKey, 130
- DSAPublicKey
 - QCA::DSAPublicKey, 132
- DTLS_v1
 - QCA::TLS, 295
- e
 - QCA::RSAPrivateKey, 243
 - QCA::RSAPublicKey, 245
- ECB
 - QCA::Cipher, 104
- Email
 - QCA, 31
- EmailLegacy
 - QCA, 31
- EmailProtection
 - QCA, 32
- EME_PKCS1_OAEP
 - QCA, 33
- EME_PKCS1v15
 - QCA, 33
- Emergency
 - QCA::Logger, 191
- EMSA1_SHA1
 - QCA, 34
- EMSA3_MD2
 - QCA, 34
- EMSA3_MD5
 - QCA, 34
- EMSA3_Raw
 - QCA, 34
- EMSA3_RIPEMD160
 - QCA, 34
- EMSA3_SHA1
 - QCA, 34
- emsa3Encode
 - QCA, 43
- EncipherOnly
 - QCA, 31
- Encode
 - QCA, 33
- encode
 - QCA::TextFilter, 291

- encodeString
 - QCA::TextFilter, 292
- encrypt
 - QCA::PublicKey, 233
- EncryptionAlgorithm
 - QCA, 33
- end
 - QCA::SecureMessage, 277
- ensureAccess
 - QCA::KeyStoreEntry, 182
- ensureAvailable
 - QCA::KeyStoreEntry, 181
- entry
 - QCA::KeyStoreEntryWatcher, 184
- entryList
 - QCA::KeyStore, 174
- entryRemoved
 - QCA::KeyStore, 176
- entryWritten
 - QCA::KeyStore, 176
- Error
 - QCA::Logger, 191
 - QCA::SASL, 248
 - QCA::SecureMessage, 272
 - QCA::TLS, 295
- error
 - QCA::SecureLayer, 268
- ErrorCertKeyMismatch
 - QCA::SecureMessage, 272
 - QCA::TLS, 296
- errorCode
 - QCA::SASL, 252
 - QCA::SecureMessage, 277
 - QCA::TLS, 301
- ErrorCrypt
 - QCA::SASL, 248
 - QCA::TLS, 296
- ErrorDecode
 - QCA, 34
- ErrorEncryptExpired
 - QCA::SecureMessage, 272
- ErrorEncryptInvalid
 - QCA::SecureMessage, 272
- ErrorEncryptUntrusted
 - QCA::SecureMessage, 272
- ErrorExpired
 - QCA, 33
- ErrorExpiredCA
 - QCA, 33
- ErrorFile
 - QCA, 34
- ErrorFormat
 - QCA::SecureMessage, 272
- ErrorHandshake
 - QCA::SASL, 248
 - QCA::TLS, 296
- ErrorInit
 - QCA::SASL, 248
 - QCA::TLS, 296
- ErrorInvalidCA
 - QCA, 32
- ErrorInvalidPurpose
 - QCA, 32
- ErrorNeedCard
 - QCA::SecureMessage, 272
- ErrorPassphrase
 - QCA, 34
 - QCA::SecureMessage, 272
- ErrorPathLengthExceeded
 - QCA, 33
- ErrorRejected
 - QCA, 32
- ErrorRevoked
 - QCA, 32
- ErrorSelfSigned
 - QCA, 32
- ErrorSignatureFailed
 - QCA, 32
- ErrorSignerExpired
 - QCA::SecureMessage, 272
 - QCA::TLS, 296
- ErrorSignerInvalid
 - QCA::SecureMessage, 272
 - QCA::TLS, 296
- ErrorUnknown
 - QCA::SecureMessage, 272
- ErrorUntrusted
 - QCA, 32
- ErrorValidityUnknown
 - QCA, 33
- Event
 - QCA::Event, 134
- EventHandler
 - QCA::EventHandler, 139
- examples/ Directory Reference, 19
- expirationDate
 - QCA::PGPKey, 212
- Expired
 - QCA::SASL, 248
- ExtendedKeyUsage
 - QCA::ConstraintType, 111
- fileName
 - QCA::Event, 136
 - QCA::FileWatch, 142
- FileWatch
 - QCA::FileWatch, 141
- fill

- QCA::SecureArray, 262
- final
 - QCA::Base64, 54
 - QCA::BufferedComputation, 63
 - QCA::Cipher, 106
 - QCA::Filter, 144
 - QCA::Hash, 148
 - QCA::Hex, 151
 - QCA::MessageAuthenticationCode, 200
- findProvider
 - QCA, 40
- fingerprint
 - QCA::PGPKey, 212
- finished
 - QCA::SecureMessage, 278
- Format
 - QCA::SecureMessage, 271
- format
 - QCA::CertificateOptions, 91
 - QCA::CertificateRequest, 98
 - QCA::SecureMessage, 273
- fromArray
 - QCA::BigInteger, 60
 - QCA::KeyBundle, 158
 - QCA::PGPKey, 213
- fromDER
 - QCA::Certificate, 71
 - QCA::CertificateRequest, 100
 - QCA::CRL, 117
 - QCA::PrivateKey, 227
 - QCA::PublicKey, 235
- fromFile
 - QCA::KeyBundle, 159
 - QCA::PGPKey, 214
- fromFlatTextFile
 - QCA::CertificateCollection, 83
- fromPEM
 - QCA::Certificate, 71
 - QCA::CertificateRequest, 101
 - QCA::CRL, 117
 - QCA::PrivateKey, 228
 - QCA::PublicKey, 235
- fromPEMFile
 - QCA::Certificate, 72
 - QCA::CertificateRequest, 101
 - QCA::CRL, 118
 - QCA::PrivateKey, 228
 - QCA::PublicKey, 235
- fromPKCS7File
 - QCA::CertificateCollection, 83
- fromString
 - QCA::BigInteger, 60
 - QCA::CertificateRequest, 101
 - QCA::KeyStoreEntry, 181
- QCA::PGPKey, 213
- g
 - QCA::DLGroup, 127
- getProperty
 - QCA, 41
- getProviderConfig
 - QCA, 41
- globalRandomProvider
 - QCA, 41
- handshaken
 - QCA::TLS, 305
- Hash
 - QCA::Hash, 146
- hash
 - QCA::Hash, 148
- hashName
 - QCA::SecureMessage, 278
- hashToString
 - QCA::Hash, 148
- havePrivate
 - QCA::SecureMessageKey, 282
- haveSecureMemory
 - QCA, 36
- haveSecureRandom
 - QCA, 37
- haveSystemStore
 - QCA, 41
- Hex
 - QCA::Hex, 150
- hexToArray
 - QCA, 43
- holdsIdentities
 - QCA::KeyStore, 175
- holdsPGPPublicKeys
 - QCA::KeyStore, 175
- holdsTrustedCertificates
 - QCA::KeyStore, 175
- HostMismatch
 - QCA::TLS, 296
- hostName
 - QCA::TLS, 299
- id
 - QCA::CertificateInfoType, 89
 - QCA::ConstraintType, 112
 - QCA::KeyStore, 174
 - QCA::KeyStoreEntry, 180
 - QCA::KeyStoreInfo, 186
- IdentityResult
 - QCA::SecureMessageSignature, 284
 - QCA::TLS, 296
- identityResult

- QCA::SecureMessageSignature, 285
- IEEE_1363
 - QCA, 34
- IETF_1024
 - QCA, 35
- IETF_1536
 - QCA, 35
- IETF_2048
 - QCA, 35
- IETF_3072
 - QCA, 35
- IETF_4096
 - QCA, 35
- IETF_6144
 - QCA, 35
- IETF_768
 - QCA, 35
- IETF_8192
 - QCA, 35
- include/ Directory Reference, 20
- include/QtCrypto/ Directory Reference, 21
- IncorporationCountry
 - QCA, 31
- IncorporationLocality
 - QCA, 31
- IncorporationState
 - QCA, 31
- info
 - QCA::CertificateOptions, 92
- infoOrdered
 - QCA::CertificateOptions, 92
- Information
 - QCA::Logger, 191
- init
 - QCA, 36
- InitializationVector
 - QCA::InitializationVector, 152, 153
- Initializer
 - QCA::Initializer, 154
- inKeyring
 - QCA::PGPKey, 212
- insertProvider
 - QCA, 38
- InvalidCertificate
 - QCA::TLS, 296
- InvalidKey
 - QCA::SecureMessageSignature, 284
- InvalidSignature
 - QCA::SecureMessageSignature, 284
- invokeMethodWithVariants
 - QCA, 44
- IPAddress
 - QCA, 31
- IPSecEndSystem
 - QCA, 32
- IPSecTunnel
 - QCA, 32
- IPSecUser
 - QCA, 32
- isAccessible
 - QCA::KeyStoreEntry, 180
- isAvailable
 - QCA::KeyStoreEntry, 180
- isBusy
 - QCA::KeyGenerator, 164
 - QCA::KeyStoreManager, 188
- isCA
 - QCA::Certificate, 70
 - QCA::CertificateOptions, 93
 - QCA::CertificateRequest, 99
- isClosable
 - QCA::SecureLayer, 266
 - QCA::TLS, 302
- isCompressed
 - QCA::TLS, 300
- isDH
 - QCA::PKey, 218
- isDSA
 - QCA::PKey, 218
- isEmpty
 - QCA::MemoryRegion, 195
 - QCA::SecureArray, 262
- isHandshaken
 - QCA::TLS, 300
- isIssuerOf
 - QCA::Certificate, 70
- isNull
 - QCA::Certificate, 67
 - QCA::CertificateRequest, 98
 - QCA::CRL, 115
 - QCA::CRLEntry, 121
 - QCA::DLGroup, 127
 - QCA::Event, 135
 - QCA::KeyBundle, 156
 - QCA::KeyStoreEntry, 180
 - QCA::KeyStoreInfo, 186
 - QCA::MemoryRegion, 195
 - QCA::PGPKey, 211
 - QCA::PKey, 218
 - QCA::SecureMessageKey, 281
- isPrivate
 - QCA::PKey, 218
- isPublic
 - QCA::PKey, 218
- isReadOnly
 - QCA::KeyStore, 174
- isRSA
 - QCA::PKey, 218

- isSecret
 - QCA::PGPKey, 212
- isSecure
 - QCA::MemoryRegion, 195
- isSelfSigned
 - QCA::Certificate, 70
- issuerInfo
 - QCA::Certificate, 68
 - QCA::CRL, 115
- issuerInfoOrdered
 - QCA::Certificate, 68
 - QCA::CRL, 115
- issuerKeyId
 - QCA::Certificate, 70
 - QCA::CRL, 116
- issuerList
 - QCA::TLS, 298
- issuerLocations
 - QCA::Certificate, 69
 - QCA::CertificateOptions, 92
- isSupported
 - QCA, 37
- isTrusted
 - QCA::PGPKey, 213
- isValid
 - QCA::CertificateOptions, 91
 - QCA::KeyStore, 174
- isWeakDESKey
 - QCA::SymmetricKey, 289
- key
 - QCA::KeyGenerator, 165
 - QCA::SecureMessageSignature, 285
- KeyAgreement
 - QCA, 31
- KeyBundle
 - QCA::KeyBundle, 156
- keyBundle
 - QCA::KeyLoader, 171
 - QCA::KeyStoreEntry, 181
- KeyCertificateSign
 - QCA, 31
- KeyCompromise
 - QCA::CRLEntry, 120
- KeyDerivationFunction
 - QCA::KeyDerivationFunction, 162
- KeyEncipherment
 - QCA, 31
- KeyGenerator
 - QCA::KeyGenerator, 164
- keyId
 - QCA::PGPKey, 212
- KeyLength
 - QCA::KeyLength, 167
- keyLength
 - QCA::Cipher, 106
 - QCA::MessageAuthenticationCode, 199
- KeyLoader
 - QCA::KeyLoader, 170
- KeyStore
 - QCA::Event, 134
 - QCA::KeyStore, 174
- keyStoreAvailable
 - QCA::KeyStoreManager, 189
- KeyStoreEntry
 - QCA::KeyStoreEntry, 179
- keyStoreEntry
 - QCA::Event, 136
- KeyStoreEntryWatcher
 - QCA::KeyStoreEntryWatcher, 184
- KeyStoreInfo
 - QCA::KeyStoreInfo, 185, 186
- keyStoreInfo
 - QCA::Event, 135
- KeyStoreManager
 - QCA::KeyStoreManager, 188
- keyStores
 - QCA::KeyStoreManager, 188
- KeyUsage
 - QCA::ConstraintType, 111
- keyValidity
 - QCA::SecureMessageSignature, 285
- known
 - QCA::CertificateInfoType, 89
 - QCA::ConstraintType, 112
- level
 - QCA::Logger, 191
- lineBreaksColumn
 - QCA::Base64, 53
- lineBreaksEnabled
 - QCA::Base64, 53
- loadKeyBundleFromArray
 - QCA::KeyLoader, 171
- loadKeyBundleFromFile
 - QCA::KeyLoader, 171
- loadPrivateKeyFromDER
 - QCA::KeyLoader, 170
- loadPrivateKeyFromPEM
 - QCA::KeyLoader, 170
- loadPrivateKeyFromPEMFile
 - QCA::KeyLoader, 170
- localCertificateChain
 - QCA::TLS, 302
- Locality
 - QCA, 31
- localPrivateKey
 - QCA::TLS, 302

- Locking
 - QCA, 33
- LockingKeepPrivileges
 - QCA, 33
- logBinaryMessage
 - QCA::AbstractLogDevice, 48
 - QCA::Logger, 192
- logger
 - QCA, 41
- logTextMessage
 - QCA::AbstractLogDevice, 48
 - QCA::Logger, 191
- makeFriendlyNames
 - QCA, 36
- makeKey
 - QCA::KeyDerivationFunction, 162
- matchesHostName
 - QCA::Certificate, 72
- maximum
 - QCA::KeyLength, 168
- maximumEncryptSize
 - QCA::PublicKey, 233
- mechanism
 - QCA::SASL, 252
- mechanismList
 - QCA::SASL, 252
- MemoryMode
 - QCA, 33
- MemoryRegion
 - QCA::MemoryRegion, 194
- Message
 - QCA::SecureMessage, 271
- MessageAuthenticationCode
 - QCA::MessageAuthenticationCode, 199
- methodReturnType
 - QCA, 43
- minimum
 - QCA::KeyLength, 168
- Mode
 - QCA::Cipher, 104
 - QCA::TLS, 295
- mode
 - QCA::Cipher, 105
- multiple
 - QCA::KeyLength, 168
- n
 - QCA::RSAPrivateKey, 243
 - QCA::RSAPublicKey, 245
- name
 - QCA::AbstractLogDevice, 48
 - QCA::KeyBundle, 156
 - QCA::KeyStore, 174
 - QCA::KeyStoreEntry, 180
 - QCA::KeyStoreInfo, 186
 - QCA::SecureMessageKey, 282
- NeedEncrypt
 - QCA::SASL, 248
- needParams
 - QCA::SASL, 255
- needPassword
 - QCA::SASL::Params, 257
- needUsername
 - QCA::SASL::Params, 257
- nextByte
 - QCA::Random, 240
- nextBytes
 - QCA::Random, 240
- nextStep
 - QCA::SASL, 255
- nextUpdate
 - QCA::CRL, 116
- NoAuthzid
 - QCA::SASL, 248
- NoCertificate
 - QCA::TLS, 296
- NoKey
 - QCA::SecureMessageSignature, 284
- NoMechanism
 - QCA::SASL, 248
- None
 - QCA::SecureMessageKey, 281
- NonRepudiation
 - QCA, 31
- NoPadding
 - QCA::Cipher, 104
- Notice
 - QCA::Logger, 191
- notValidAfter
 - QCA::Certificate, 67
 - QCA::CertificateOptions, 93
- notValidBefore
 - QCA::Certificate, 67
 - QCA::CertificateOptions, 93
- NoUser
 - QCA::SASL, 248
- number
 - QCA::CRL, 116
- ocspLocations
 - QCA::Certificate, 69
 - QCA::CertificateOptions, 93
- OCSPSigning
 - QCA, 32
- OFB
 - QCA::Cipher, 104
- ok

- QCA::Base64, 54
- QCA::Cipher, 106
- QCA::Filter, 144
- QCA::Hex, 151
- OpenPGP
 - QCA::OpenPGP, 201
 - QCA::SecureMessage, 271
- operator!=
 - QCA::BigInteger, 61
 - QCA::Certificate, 72
 - QCA::CertificateInfoPair, 86
 - QCA::CertificateInfoType, 89
 - QCA::CertificateRequest, 100
 - QCA::ConstraintType, 113
 - QCA::CRL, 116
 - QCA::CRLEntry, 121
 - QCA::PKey, 219
 - QCA::SecureArray, 263
- operator+
 - QCA, 45
 - QCA::CertificateCollection, 82
- operator+=
 - QCA::BigInteger, 59
 - QCA::CertificateCollection, 82
 - QCA::SecureArray, 263
- operator-=
 - QCA::BigInteger, 59
- operator<
 - QCA::BigInteger, 61
 - QCA::CertificateInfoType, 89
 - QCA::ConstraintType, 113
 - QCA::CRLEntry, 121
- operator<<
 - QCA::BigInteger, 62
- operator<=
 - QCA::BigInteger, 61
- operator=
 - QCA::Algorithm, 50
 - QCA::BigInteger, 59
 - QCA::Certificate, 67
 - QCA::CertificateAuthority, 75
 - QCA::CertificateCollection, 81
 - QCA::CertificateInfoPair, 86
 - QCA::CertificateInfoType, 88
 - QCA::CertificateOptions, 91
 - QCA::CertificateRequest, 98
 - QCA::Cipher, 105
 - QCA::ConstraintType, 112
 - QCA::CRL, 115
 - QCA::CRLEntry, 121
 - QCA::DLGroup, 127
 - QCA::Event, 135
 - QCA::Hash, 146
 - QCA::KeyBundle, 156
 - QCA::KeyDerivationFunction, 162
 - QCA::KeyStoreEntry, 180
 - QCA::KeyStoreInfo, 186
 - QCA::MemoryRegion, 195
 - QCA::MessageAuthenticationCode, 199
 - QCA::PGPKey, 211
 - QCA::PKey, 217
 - QCA::PrivateKey, 224
 - QCA::PublicKey, 232
 - QCA::Random, 240
 - QCA::SASL::Params, 257
 - QCA::SecureArray, 260
 - QCA::SecureMessageKey, 281
 - QCA::SecureMessageSignature, 285
- operator==
 - QCA::BigInteger, 61
 - QCA::Certificate, 72
 - QCA::CertificateInfoPair, 86
 - QCA::CertificateInfoType, 89
 - QCA::CertificateRequest, 99
 - QCA::ConstraintType, 113
 - QCA::CRL, 116
 - QCA::CRLEntry, 121
 - QCA::PKey, 219
 - QCA::SecureArray, 263
- operator>
 - QCA::BigInteger, 62
- operator>=
 - QCA::BigInteger, 61
- operator[]
 - QCA::SecureArray, 260
- orderedDNOnly
 - QCA, 36
- orderedToDNString
 - QCA, 36
- Organization
 - QCA, 31
- OrganizationalUnit
 - QCA, 31
- p
 - QCA::DLGroup, 127
 - QCA::RSAPrivateKey, 243
- packetMTU
 - QCA::TLS, 304
- packetsAvailable
 - QCA::TLS, 304
- packetsOutgoingAvailable
 - QCA::TLS, 304
- Padding
 - QCA::Cipher, 104
- padding
 - QCA::Cipher, 105
- Params

- QCA::SASL::Params, 256
- Password
 - QCA::Event, 134
- password
 - QCA::PasswordAsker, 205
- PasswordAsker
 - QCA::PasswordAsker, 204
- PasswordStyle
 - QCA::Event, 134
- passwordStyle
 - QCA::Event, 135
- pathLimit
 - QCA::Certificate, 70
 - QCA::CertificateOptions, 93
 - QCA::CertificateRequest, 99
- PBEAlgorithm
 - QCA, 34
- PBEDefault
 - QCA, 34
- PBES2_AES128_SHA1
 - QCA, 34
- PBES2_AES192_SHA1
 - QCA, 34
- PBES2_AES256_SHA1
 - QCA, 34
- PBES2_DES_SHA1
 - QCA, 34
- PBES2_TripleDES_SHA1
 - QCA, 34
- PBKDF1
 - QCA::PBKDF1, 206
- PBKDF2
 - QCA::PBKDF2, 208
- peerCertificateAvailable
 - QCA::TLS, 305
- peerCertificateChain
 - QCA::TLS, 302
- peerCertificateValidity
 - QCA::TLS, 302
- peerIdentityResult
 - QCA::TLS, 302
- PGP
 - QCA::SecureMessageKey, 281
- PGPKey
 - QCA::PGPKey, 211
- PGPKeyring
 - QCA::KeyStore, 173
- pgpPublicKey
 - QCA::KeyStoreEntry, 181
 - QCA::SecureMessageKey, 281
- pgpSecretKey
 - QCA::KeyStoreEntry, 181
 - QCA::SecureMessageKey, 281
- PKCS10
 - QCA, 30
- PKCS7
 - QCA::Cipher, 104
- PKey
 - QCA::PKey, 216
- pluginDiagnosticText
 - QCA, 40
- policies
 - QCA::Certificate, 69
 - QCA::CertificateOptions, 92
 - QCA::CertificateRequest, 99
- Practical
 - QCA, 33
- primary
 - QCA::CertificateChain, 78
- primaryUserId
 - QCA::PGPKey, 212
- PrivateKey
 - QCA::PrivateKey, 223, 224
- privateKey
 - QCA::KeyBundle, 157
 - QCA::KeyLoader, 171
- privateKeys
 - QCA::CMS, 109
- process
 - QCA::BufferedComputation, 63
 - QCA::Filter, 144
- provider
 - QCA::Algorithm, 51
- ProviderList
 - QCA, 30
- providerPriority
 - QCA, 39
- providers
 - QCA, 40
- ptr
 - QCA::Event, 136
- PublicKey
 - QCA::PublicKey, 231, 232
- putServerFirstStep
 - QCA::SASL, 251
- putStep
 - QCA::SASL, 251
- q
 - QCA::DLGroup, 127
 - QCA::RSAPrivateKey, 243
- QCA, 23
 - appendPluginDiagnosticText, 40
 - appName, 42
 - arrayToHex, 42
 - CertificateInfo, 30
 - CertificateInfoTypeKnown, 30
 - CertificateRequestFormat, 30

- clearPluginDiagnosticText, 40
- ClientAuth, 32
- CodeSigning, 32
- CommonName, 31
- Constraints, 30
- ConstraintTypeKnown, 31
- ConvertGood, 34
- ConvertResult, 34
- Country, 31
- CRLSign, 31
- DataEncipherment, 31
- DecipherOnly, 31
- Decode, 33
- defaultFeatures, 38
- DefaultFormat, 34
- defaultProvider, 40
- deinit, 36
- DERSequence, 34
- DigitalSignature, 31
- Direction, 33
- DLGroupSet, 34
- DNS, 31
- DSA_1024, 35
- DSA_512, 35
- DSA_768, 35
- Email, 31
- EmailLegacy, 31
- EmailProtection, 32
- EME_PKCS1_OAEP, 33
- EME_PKCS1v15, 33
- EMSA1_SHA1, 34
- EMSA3_MD2, 34
- EMSA3_MD5, 34
- EMSA3_Raw, 34
- EMSA3_RIPEMD160, 34
- EMSA3_SHA1, 34
- emsa3Encode, 43
- EncipherOnly, 31
- Encode, 33
- EncryptionAlgorithm, 33
- ErrorDecode, 34
- ErrorExpired, 33
- ErrorExpiredCA, 33
- ErrorFile, 34
- ErrorInvalidCA, 32
- ErrorInvalidPurpose, 32
- ErrorPassphrase, 34
- ErrorPathLengthExceeded, 33
- ErrorRejected, 32
- ErrorRevoked, 32
- ErrorSelfSigned, 32
- ErrorSignatureFailed, 32
- ErrorUntrusted, 32
- ErrorValidityUnknown, 33
- findProvider, 40
- getProperty, 41
- getProviderConfig, 41
- globalRandomProvider, 41
- haveSecureMemory, 36
- haveSecureRandom, 37
- haveSystemStore, 41
- hexToArray, 43
- IEEE_1363, 34
- IETF_1024, 35
- IETF_1536, 35
- IETF_2048, 35
- IETF_3072, 35
- IETF_4096, 35
- IETF_6144, 35
- IETF_768, 35
- IETF_8192, 35
- IncorporationCountry, 31
- IncorporationLocality, 31
- IncorporationState, 31
- init, 36
- insertProvider, 38
- invokeMethodWithVariants, 44
- IPAddress, 31
- IPSecEndSystem, 32
- IPSecTunnel, 32
- IPSecUser, 32
- isSupported, 37
- KeyAgreement, 31
- KeyCertificateSign, 31
- KeyEncipherment, 31
- Locality, 31
- Locking, 33
- LockingKeepPrivileges, 33
- logger, 41
- makeFriendlyNames, 36
- MemoryMode, 33
- methodReturnType, 43
- NonRepudiation, 31
- OCSPSigning, 32
- operator+, 45
- orderedDNOnly, 36
- orderedToDNString, 36
- Organization, 31
- OrganizationalUnit, 31
- PBEAlgorithm, 34
- PBEDefault, 34
- PBES2_AES128_SHA1, 34
- PBES2_AES192_SHA1, 34
- PBES2_AES256_SHA1, 34
- PBES2_DES_SHA1, 34
- PBES2_TripleDES_SHA1, 34
- PKCS10, 30
- pluginDiagnosticText, 40

- Practical, 33
- ProviderList, 30
- providerPriority, 39
- providers, 40
- saveProviderConfig, 41
- scanForPlugins, 40
- SecureMessageKeyList, 30
- SecureMessageSignatureList, 30
- SecurityLevel, 35
- ServerAuth, 31
- setAppName, 42
- setGlobalRandomProvider, 41
- setProperty, 41
- setProviderConfig, 41
- setProviderPriority, 39
- SignatureAlgorithm, 33
- SignatureFormat, 34
- SignatureUnknown, 34
- SL_Baseline, 35
- SL_Export, 35
- SL_High, 35
- SL_Highest, 35
- SL_Integrity, 35
- SL_None, 35
- SPKAC, 30
- State, 31
- supportedFeatures, 38
- systemStore, 42
- TimeStamping, 32
- unloadAllPlugins, 40
- URI, 31
- UsageAny, 32
- UsageCodeSigning, 32
- UsageCRLSigning, 32
- UsageEmailProtection, 32
- UsageMode, 32
- UsageTimeStamping, 32
- UsageTLSClient, 32
- UsageTLSServer, 32
- ValidateFlags, 33
- Validity, 32
- ValidityGood, 32
- XMPP, 31
- qca.h, 309
- QCA::AbstractLogDevice, 47
- QCA::AbstractLogDevice
 - AbstractLogDevice, 48
 - logBinaryMessage, 48
 - logTextMessage, 48
 - name, 48
- QCA::Algorithm, 49
 - Algorithm, 50
 - change, 51
 - context, 51
 - operator=, 50
 - provider, 51
 - takeContext, 51
 - type, 50
- QCA::Base64, 52
 - Base64, 53
 - clear, 53
 - final, 54
 - lineBreaksColumn, 53
 - lineBreaksEnabled, 53
 - ok, 54
 - setLineBreaksColumn, 53
 - setLineBreaksEnabled, 53
 - update, 53
- QCA::BasicContext, 55
- QCA::BasicContext
 - BasicContext, 55
- QCA::BigInteger, 57
- QCA::BigInteger
 - BigInteger, 58
 - compare, 61
 - fromArray, 60
 - fromString, 60
 - operator!=, 61
 - operator+=, 59
 - operator=, 59
 - operator<, 61
 - operator<<, 62
 - operator<=, 61
 - operator=, 59
 - operator==, 61
 - operator>, 62
 - operator>=, 61
 - toArray, 60
 - toString, 60
- QCA::BufferedComputation, 63
- QCA::BufferedComputation
 - clear, 63
 - final, 63
 - process, 63
 - update, 63
- QCA::Certificate, 65
 - Certificate, 66, 67
 - change, 73
 - commonName, 69
 - constraints, 69
 - crlLocations, 69
 - fromDER, 71
 - fromPEM, 71
 - fromPEMFile, 72
 - isCA, 70
 - isIssuerOf, 70
 - isNull, 67
 - isSelfSigned, 70

- issuerInfo, 68
- issuerInfoOrdered, 68
- issuerKeyId, 70
- issuerLocations, 69
- matchesHostName, 72
- notValidAfter, 67
- notValidBefore, 67
- ocspLocations, 69
- operator!=, 72
- operator=, 67
- operator==, 72
- pathLimit, 70
- policies, 69
- serialNumber, 69
- signatureAlgorithm, 70
- subjectInfo, 67
- subjectInfoOrdered, 68
- subjectKeyId, 70
- subjectPublicKey, 69
- toDER, 71
- toPEM, 71
- toPEMFile, 71
- validate, 70
- QCA::CertificateAuthority, 74
- QCA::CertificateAuthority
 - certificate, 75
 - CertificateAuthority, 74, 75
 - createCertificate, 75
 - createCRL, 75
 - operator=, 75
 - signRequest, 75
 - updateCRL, 76
- QCA::CertificateChain, 77
- QCA::CertificateChain
 - CertificateChain, 78
 - complete, 78
 - primary, 78
 - validate, 78
- QCA::CertificateCollection, 80
- QCA::CertificateCollection
 - addCertificate, 81
 - addCRL, 81
 - append, 82
 - canUsePKCS7, 82
 - CertificateCollection, 81
 - certificates, 81
 - crls, 81
 - fromFlatTextFile, 83
 - fromPKCS7File, 83
 - operator+, 82
 - operator+=, 82
 - operator=, 81
 - toFlatTextFile, 82
 - toPKCS7File, 82
 - QCA::CertificateInfoOrdered, 84
 - QCA::CertificateInfoOrdered
 - dnOnly, 84
 - toString, 84
 - QCA::CertificateInfoPair, 85
 - QCA::CertificateInfoPair
 - CertificateInfoPair, 85
 - operator!=, 86
 - operator=, 86
 - operator==, 86
 - type, 86
 - value, 86
 - QCA::CertificateInfoType, 87
 - AlternativeName, 88
 - DN, 88
 - QCA::CertificateInfoType
 - CertificateInfoType, 88
 - id, 89
 - known, 89
 - operator!=, 89
 - operator<, 89
 - operator=, 88
 - operator==, 89
 - Section, 88
 - section, 88
 - QCA::CertificateOptions, 90
 - QCA::CertificateOptions
 - CertificateOptions, 91
 - challenge, 92
 - constraints, 92
 - crlLocations, 92
 - format, 91
 - info, 92
 - infoOrdered, 92
 - isCA, 93
 - issuerLocations, 92
 - isValid, 91
 - notValidAfter, 93
 - notValidBefore, 93
 - ocspLocations, 93
 - operator=, 91
 - pathLimit, 93
 - policies, 92
 - serialNumber, 93
 - setAsCA, 95
 - setAsUser, 95
 - setChallenge, 93
 - setConstraints, 94
 - setCRLLocations, 94
 - setFormat, 91
 - setInfo, 94
 - setInfoOrdered, 94
 - setIssuerLocations, 94
 - setOCSPLocations, 95

- setPolicies, 94
- setSerialNumber, 95
- setValidityPeriod, 95
- QCA::CertificateRequest, 96
- QCA::CertificateRequest
 - canUseFormat, 98
 - CertificateRequest, 97
 - challenge, 99
 - change, 102
 - constraints, 99
 - format, 98
 - fromDER, 100
 - fromPEM, 101
 - fromPEMFile, 101
 - fromString, 101
 - isCA, 99
 - isNull, 98
 - operator!=, 100
 - operator=, 98
 - operator==, 99
 - pathLimit, 99
 - policies, 99
 - signatureAlgorithm, 99
 - subjectInfo, 98
 - subjectInfoOrdered, 98
 - subjectPublicKey, 99
 - toDER, 100
 - toPEM, 100
 - toPEMFile, 100
 - toString, 101
- QCA::Cipher, 103
 - blockSize, 106
 - CBC, 104
 - CFB, 104
 - Cipher, 105
 - clear, 106
 - DefaultPadding, 104
 - direction, 105
 - ECB, 104
 - final, 106
 - keyLength, 106
 - Mode, 104
 - mode, 105
 - NoPadding, 104
 - OFB, 104
 - ok, 106
 - operator=, 105
 - Padding, 104
 - padding, 105
 - PKCS7, 104
 - setup, 107
 - type, 105
 - update, 106
 - validKeyLength, 106
 - withAlgorithms, 107
- QCA::CMS, 108
 - CMS, 109
 - privateKeys, 109
 - setPrivateKeys, 109
 - setTrustedCertificates, 109
 - setUntrustedCertificates, 109
 - trustedCertificates, 109
 - untrustedCertificates, 109
- QCA::ConstraintType, 111
 - ExtendedKeyUsage, 111
 - KeyUsage, 111
- QCA::ConstraintType
 - ConstraintType, 112
 - id, 112
 - known, 112
 - operator!=, 113
 - operator<, 113
 - operator=, 112
 - operator==, 113
 - Section, 111
 - section, 112
- QCA::CRL, 114
 - change, 118
 - CRL, 115
 - fromDER, 117
 - fromPEM, 117
 - fromPEMFile, 118
 - isNull, 115
 - issuerInfo, 115
 - issuerInfoOrdered, 115
 - issuerKeyId, 116
 - nextUpdate, 116
 - number, 116
 - operator!=, 116
 - operator=, 115
 - operator==, 116
 - revoked, 116
 - signatureAlgorithm, 116
 - thisUpdate, 116
 - toDER, 117
 - toPEM, 117
 - toPEMFile, 117
- QCA::CRLEntry, 119
 - AACompromise, 120
 - CACompromise, 120
 - CertificateHold, 120
 - CRLEntry, 120
 - isNull, 121
 - KeyCompromise, 120
 - operator!=, 121
 - operator<, 121
 - operator=, 121
 - operator==, 121

- Reason, 119
- reason, 121
- RemoveFromCRL, 120
- serialNumber, 121
- Superseded, 120
- time, 121
- Unspecified, 120
- QCA::DHPrivateKey, 122
- QCA::DHPrivateKey
 - DHPrivateKey, 123
 - domain, 123
 - x, 123
 - y, 123
- QCA::DHPublicKey, 124
- QCA::DHPublicKey
 - DHPublicKey, 125
 - domain, 125
 - y, 125
- QCA::DLGroup, 126
- DLGroup, 126, 127
- g, 127
- isNull, 127
- operator=, 127
- p, 127
- q, 127
- supportedGroupSets, 127
- QCA::DSAPrivateKey, 129
- QCA::DSAPrivateKey
 - domain, 130
 - DSAPrivateKey, 130
 - x, 130
 - y, 130
- QCA::DSAPublicKey, 131
- QCA::DSAPublicKey
 - domain, 132
 - DSAPublicKey, 132
 - y, 132
- QCA::Event, 133
- ~Event, 135
- Data, 134
- Event, 134
- fileName, 136
- isNull, 135
- KeyStore, 134
- keyStoreEntry, 136
- keyStoreInfo, 135
- operator=, 135
- Password, 134
- PasswordStyle, 134
- passwordStyle, 135
- ptr, 136
- setPasswordData, 136
- setPasswordKeyStore, 136
- setToken, 136
- Source, 134
- source, 135
- StylePassphrase, 134
- StylePassword, 134
- StylePIN, 134
- Token, 134
- Type, 134
- type, 135
- QCA::EventHandler, 138
- QCA::EventHandler
 - EventHandler, 139
 - reject, 139
 - start, 139
 - submitPassword, 139
 - tokenOkay, 139
- QCA::FileWatch, 141
- QCA::FileWatch
 - fileName, 142
 - FileWatch, 141
 - setFileName, 142
- QCA::Filter, 143
- clear, 143
- final, 144
- ok, 144
- process, 144
- update, 143
- QCA::Hash, 145
- clear, 147
- final, 148
- Hash, 146
- hash, 148
- hashToString, 148
- operator=, 146
- type, 146
- update, 147
- QCA::Hex, 150
- clear, 151
- final, 151
- Hex, 150
- ok, 151
- update, 151
- QCA::InitializationVector, 152
- QCA::InitializationVector
 - InitializationVector, 152, 153
- QCA::Initializer, 154
- Initializer, 154
- QCA::KeyBundle, 155
- QCA::KeyBundle
 - certificateChain, 157
 - fromArray, 158
 - fromFile, 159
 - isNull, 156
 - KeyBundle, 156
 - name, 156

- operator=, 156
- privateKey, 157
- setCertificateChainAndKey, 157
- setName, 157
- toArray, 157
- toFile, 158
- QCA::KeyDerivationFunction, 161
- QCA::KeyDerivationFunction
 - KeyDerivationFunction, 162
 - makeKey, 162
 - operator=, 162
 - withAlgorithm, 162
- QCA::KeyGenerator, 163
- QCA::KeyGenerator
 - blockingEnabled, 164
 - createDH, 165
 - createDLGroup, 165
 - createDSA, 165
 - createRSA, 164
 - dlGroup, 165
 - isBusy, 164
 - key, 165
 - KeyGenerator, 164
 - setBlockingEnabled, 164
- QCA::KeyLength, 167
- QCA::KeyLength
 - KeyLength, 167
 - maximum, 168
 - minimum, 168
 - multiple, 168
- QCA::KeyLoader, 169
- QCA::KeyLoader
 - convertResult, 171
 - keyBundle, 171
 - KeyLoader, 170
 - loadKeyBundleFromArray, 171
 - loadKeyBundleFromFile, 171
 - loadPrivateKeyFromDER, 170
 - loadPrivateKeyFromPEM, 170
 - loadPrivateKeyFromPEMFile, 170
 - privateKey, 171
- QCA::KeyStore, 172
 - Application, 173
 - PGPKeyring, 173
 - SmartCard, 173
 - System, 173
 - User, 173
- QCA::KeyStore
 - entryList, 174
 - entryRemoved, 176
 - entryWritten, 176
 - holdsIdentities, 175
 - holdsPGPPublicKeys, 175
 - holdsTrustedCertificates, 175
 - id, 174
 - isReadOnly, 174
 - isValid, 174
 - KeyStore, 174
 - name, 174
 - removeEntry, 176
 - startAsynchronousMode, 174
 - Type, 173
 - type, 174
 - unavailable, 176
 - writeEntry, 175, 176
- QCA::KeyStoreEntry, 177
- QCA::KeyStoreEntry
 - certificate, 181
 - crl, 181
 - ensureAccess, 182
 - ensureAvailable, 181
 - fromString, 181
 - id, 180
 - isAccessible, 180
 - isAvailable, 180
 - isNull, 180
 - keyBundle, 181
 - KeyStoreEntry, 179
 - name, 180
 - operator=, 180
 - pgpPublicKey, 181
 - pgpSecretKey, 181
 - storeId, 181
 - storeName, 181
 - toString, 181
 - Type, 179
 - type, 180
- QCA::KeyStoreEntryWatcher, 183
- QCA::KeyStoreEntryWatcher
 - entry, 184
 - KeyStoreEntryWatcher, 184
 - unavailable, 184
- QCA::KeyStoreInfo, 185
- QCA::KeyStoreInfo
 - id, 186
 - isNull, 186
 - KeyStoreInfo, 185, 186
 - name, 186
 - operator=, 186
 - type, 186
- QCA::KeyStoreManager, 187
- QCA::KeyStoreManager
 - busyFinished, 189
 - clearDiagnosticText, 188
 - diagnosticText, 188
 - isBusy, 188
 - keyStoreAvailable, 189
 - KeyStoreManager, 188

- keyStores, 188
- start, 188
- sync, 188
- waitForBusyFinished, 188
- QCA::Logger, 190
 - Alert, 191
 - Critical, 191
 - currentLogDevices, 192
 - Debug, 191
 - Emergency, 191
 - Error, 191
 - Information, 191
 - level, 191
 - logBinaryMessage, 192
 - logTextMessage, 191
 - Notice, 191
 - Quiet, 191
 - registerLogDevice, 192
 - setLevel, 191
 - Severity, 191
 - unregisterLogDevice, 192
 - Warning, 191
- QCA::MemoryRegion, 193
- QCA::MemoryRegion
 - at, 196, 197
 - constData, 196
 - data, 196
 - isEmpty, 195
 - isNull, 195
 - isSecure, 195
 - MemoryRegion, 194
 - operator=, 195
 - resize, 197
 - set, 197
 - setSecure, 197
 - size, 196
 - toByteArray, 195
- QCA::MessageAuthenticationCode, 198
- QCA::MessageAuthenticationCode
 - clear, 199
 - final, 200
 - keyLength, 199
 - MessageAuthenticationCode, 199
 - operator=, 199
 - setup, 200
 - type, 199
 - update, 200
 - validKeyLength, 199
- QCA::OpenPGP, 201
- QCA::OpenPGP
 - OpenPGP, 201
- QCA::PasswordAsker, 203
- QCA::PasswordAsker
 - accepted, 204
 - ask, 204
 - cancel, 204
 - password, 205
 - PasswordAsker, 204
 - waitForResponse, 204
- QCA::PBKDF1, 206
 - PBKDF1, 206
- QCA::PBKDF2, 208
 - PBKDF2, 208
- QCA::PGPKey, 210
 - creationDate, 212
 - expirationDate, 212
 - fingerprint, 212
 - fromArray, 213
 - fromFile, 214
 - fromString, 213
 - inKeyring, 212
 - isNull, 211
 - isSecret, 212
 - isTrusted, 213
 - keyId, 212
 - operator=, 211
 - PGPKey, 211
 - primaryUserId, 212
 - toArray, 213
 - toFile, 213
 - toString, 213
 - userIds, 212
- QCA::PKey, 215
 - bitSize, 218
 - canExport, 219
 - canKeyAgree, 219
 - DH, 216
 - DSA, 216
 - isDH, 218
 - isDSA, 218
 - isNull, 218
 - isPrivate, 218
 - isPublic, 218
 - isRSA, 218
 - operator!=, 219
 - operator=, 217
 - operator==, 219
 - PKey, 216
 - RSA, 216
 - set, 219
 - supportedIOTypes, 217
 - supportedTypes, 217
 - toDHPrivateKey, 220
 - toDHPublicKey, 220
 - toDSAPrivateKey, 220
 - toDSAPublicKey, 220
 - toPrivateKey, 219
 - toPublicKey, 219

- toRSAPrivateKey, 219
- toRSAPublicKey, 219
- Type, 216
- type, 218
- QCA::PrivateKey, 222
- QCA::PrivateKey
 - canDecrypt, 224
 - canSign, 224
 - decrypt, 225
 - deriveKey, 226
 - fromDER, 227
 - fromPEM, 228
 - fromPEMFile, 228
 - operator=, 224
 - PrivateKey, 223, 224
 - signature, 226
 - signMessage, 226
 - startSign, 225
 - supportedPBEAlgorithms, 226
 - toDER, 226
 - toDH, 224
 - toDSA, 224
 - toPEM, 227
 - toPEMFile, 227
 - toRSA, 224
 - update, 225
- QCA::PublicKey, 230
- QCA::PublicKey
 - canEncrypt, 232
 - canVerify, 233
 - encrypt, 233
 - fromDER, 235
 - fromPEM, 235
 - fromPEMFile, 235
 - maximumEncryptSize, 233
 - operator=, 232
 - PublicKey, 231, 232
 - startVerify, 233
 - toDER, 234
 - toDH, 232
 - toDSA, 232
 - toPEM, 234
 - toPEMFile, 234
 - toRSA, 232
 - update, 233
 - validSignature, 233
 - verifyMessage, 234
- QCA::QPipe, 237
 - create, 237
 - QPipe, 237
 - readEnd, 237
 - reset, 237
 - writeEnd, 238
- QCA::Random, 239
 - nextByte, 240
 - nextBytes, 240
 - operator=, 240
 - Random, 240
 - randomArray, 241
 - randomChar, 240
 - randomInt, 241
- QCA::RSAPrivateKey, 242
- QCA::RSAPrivateKey
 - d, 243
 - e, 243
 - n, 243
 - p, 243
 - q, 243
 - RSAPrivateKey, 243
- QCA::RSAPublicKey, 244
- QCA::RSAPublicKey
 - e, 245
 - n, 245
 - RSAPublicKey, 245
- QCA::SASL, 246
 - authCheck, 255
 - AuthCondition, 248
 - authCondition, 252
 - authenticated, 255
 - AuthFail, 248
 - AuthFlags, 248
 - BadAuth, 248
 - BadProtocol, 248
 - BadServer, 248
 - bytesAvailable, 254
 - bytesOutgoingAvailable, 254
 - ClientSendMode, 249
 - continueAfterAuthCheck, 253
 - continueAfterParams, 253
 - convertBytesWritten, 255
 - Disabled, 248
 - Error, 248
 - errorCode, 252
 - ErrorCrypt, 248
 - ErrorHandshake, 248
 - ErrorInit, 248
 - Expired, 248
 - mechanism, 252
 - mechanismList, 252
 - NeedEncrypt, 248
 - needParams, 255
 - nextStep, 255
 - NoAuthzid, 248
 - NoMechanism, 248
 - NoUser, 248
 - putServerFirstStep, 251
 - putStep, 251
 - read, 254

- readOutgoing, 254
- realmList, 252
- RemoteUnavailable, 248
- reset, 249
- SASL, 249
- ServerSendMode, 249
- serverStarted, 255
- setAuthzid, 253
- setConstraints, 249
- setExternalAuthId, 250
- setExternalSSF, 250
- setLocalAddress, 250
- setPassword, 253
- setRealm, 253
- setRemoteAddress, 250
- setUsername, 253
- ssf, 252
- startClient, 250
- startServer, 251
- TooWeak, 248
- write, 254
- writeIncoming, 254
- QCA::SASL::Params, 256
 - canSendAuthzid, 257
 - canSendRealm, 257
 - needPassword, 257
 - needUsername, 257
 - operator=, 257
 - Params, 256
- QCA::SecureArray, 258
- QCA::SecureArray
 - append, 263
 - at, 261
 - clear, 260
 - constData, 261
 - data, 261
 - fill, 262
 - isEmpty, 262
 - operator!=, 263
 - operator+=, 263
 - operator=, 260
 - operator==, 263
 - operator[], 260
 - resize, 262
 - SecureArray, 259, 260
 - set, 263
 - size, 262
 - toByteArray, 263
- QCA::SecureLayer, 265
- QCA::SecureLayer
 - bytesAvailable, 266
 - bytesOutgoingAvailable, 266
 - close, 266
 - closed, 268
 - convertBytesWritten, 267
 - error, 268
 - isClosable, 266
 - read, 267
 - readOutgoing, 267
 - readUnprocessed, 267
 - readyReadOutgoing, 268
 - SecureLayer, 266
 - write, 267
 - writeIncoming, 267
- QCA::SecureMessage, 269
 - Ascii, 272
 - Binary, 272
 - Clearsign, 271
 - CMS, 271
 - Detached, 271
 - ErrorCertKeyMismatch, 272
 - ErrorEncryptExpired, 272
 - ErrorEncryptInvalid, 272
 - ErrorEncryptUntrusted, 272
 - ErrorFormat, 272
 - ErrorNeedCard, 272
 - ErrorPassphrase, 272
 - ErrorSignerExpired, 272
 - ErrorSignerInvalid, 272
 - ErrorUnknown, 272
 - Message, 271
 - OpenPGP, 271
- QCA::SecureMessage
 - bundleSignerEnabled, 273
 - bytesAvailable, 276
 - bytesWritten, 278
 - canClearsign, 273
 - canSignAndEncrypt, 273
 - canSignMultiple, 272
 - diagnosticText, 278
 - end, 277
 - Error, 272
 - errorCode, 277
 - finished, 278
 - Format, 271
 - format, 273
 - hashName, 278
 - read, 276
 - recipientKeys, 273
 - reset, 273
 - SecureMessage, 272
 - setBundleSignerEnabled, 274
 - setFormat, 274
 - setRecipient, 274
 - setRecipients, 274
 - setSigner, 274
 - setSigners, 275
 - setSMIMEAttributesEnabled, 274

- signature, 277
- signer, 278
- signerKeys, 273
- signers, 278
- SignMode, 271
- smimeAttributesEnabled, 273
- startDecrypt, 275
- startEncrypt, 275
- startSign, 275
- startSignAndEncrypt, 276
- startVerify, 276
- success, 277
- Type, 271
- type, 272
- update, 276
- verifySuccess, 278
- waitForFinished, 277
- wasSigned, 278
- QCA::SecureMessageKey, 280
 - None, 281
 - PGP, 281
 - X509, 281
- QCA::SecureMessageKey
 - havePrivate, 282
 - isNull, 281
 - name, 282
 - operator=, 281
 - pgpPublicKey, 281
 - pgpSecretKey, 281
 - SecureMessageKey, 281
 - setPGPPublicKey, 281
 - setPGPSecretKey, 282
 - setX509CertificateChain, 282
 - setX509KeyBundle, 282
 - setX509PrivateKey, 282
 - Type, 280
 - type, 281
 - x509CertificateChain, 282
 - x509PrivateKey, 282
- QCA::SecureMessageSignature, 284
 - InvalidKey, 284
 - InvalidSignature, 284
 - NoKey, 284
 - Valid, 284
- QCA::SecureMessageSignature
 - IdentityResult, 284
 - identityResult, 285
 - key, 285
 - keyValidity, 285
 - operator=, 285
 - SecureMessageSignature, 285
 - timestamp, 285
- QCA::SecureMessageSystem, 286
- QCA::SecureMessageSystem
 - SecureMessageSystem, 286
 - QCA::SymmetricKey, 288
 - QCA::SymmetricKey
 - isWeakDESKey, 289
 - SymmetricKey, 288, 289
 - QCA::TextFilter, 290
 - QCA::TextFilter
 - _dir, 292
 - arrayToString, 291
 - decode, 291
 - decodeString, 292
 - direction, 291
 - encode, 291
 - encodeString, 292
 - setup, 291
 - stringToArray, 292
 - TextFilter, 291
 - QCA::TLS, 293
 - ~TLS, 296
 - bytesAvailable, 302
 - bytesOutgoingAvailable, 303
 - canCompress, 299
 - canSetHostName, 299
 - certificateRequested, 305
 - cipherBits, 301
 - cipherMaxBits, 301
 - cipherSuite, 301
 - close, 303
 - compressionEnabled, 299
 - continueAfterStep, 300
 - convertBytesWritten, 304
 - Datagram, 295
 - DTLS_v1, 295
 - Error, 295
 - ErrorCertKeyMismatch, 296
 - errorCode, 301
 - ErrorCrypt, 296
 - ErrorHandshake, 296
 - ErrorInit, 296
 - ErrorSignerExpired, 296
 - ErrorSignerInvalid, 296
 - handshaken, 305
 - HostMismatch, 296
 - hostName, 299
 - IdentityResult, 296
 - InvalidCertificate, 296
 - isClosable, 302
 - isCompressed, 300
 - isHandshaken, 300
 - issuerList, 298
 - localCertificateChain, 302
 - localPrivateKey, 302
 - Mode, 295
 - NoCertificate, 296

- packetMTU, 304
- packetsAvailable, 304
- packetsOutgoingAvailable, 304
- peerCertificateAvailable, 305
- peerCertificateChain, 302
- peerCertificateValidity, 302
- peerIdentityResult, 302
- read, 303
- readOutgoing, 304
- readUnprocessed, 304
- reset, 297
- session, 301
- setCertificate, 297
- setCompressionEnabled, 299
- setConstraints, 298
- setIssuerList, 299
- setPacketMTU, 304
- setSession, 299
- setTrustedCertificates, 297
- SSL_v2, 295
- SSL_v3, 295
- startClient, 300
- startServer, 300
- Stream, 295
- supportedCipherSuites, 297
- TLS, 296
- TLS_v1, 295
- trustedCertificates, 297
- Valid, 296
- Version, 295
- version, 301
- write, 303
- writeIncoming, 303
- QCA::TLSSession, 306
- QCA::TokenAsker, 307
- QCA::TokenAsker
 - accepted, 308
 - ask, 308
 - cancel, 308
 - TokenAsker, 308
 - waitForResponse, 308
- qca_basic.h, 311
- qca_cert.h, 313
- qca_core.h, 316
 - QCA_logBinaryMessage, 320
 - QCA_logTextMessage, 319
 - QCA_VERSION, 319
 - qcaVersion, 320
- qca_export.h, 321
- qca_keystore.h, 322
- QCA_logBinaryMessage
 - qca_core.h, 320
- QCA_logTextMessage
 - qca_core.h, 319
- qca_publickey.h, 323
- qca_secure_alloc
 - qca_tools.h, 336
- qca_secure_free
 - qca_tools.h, 336
- qca_secure_realloc
 - qca_tools.h, 336
- qca_securelayer.h, 326
- qca_securemessage.h, 328
- qca_support.h, 330
- qca_textfilter.h, 333
- qca_tools.h, 335
 - qca_secure_alloc, 336
 - qca_secure_free, 336
 - qca_secure_realloc, 336
- QCA_VERSION
 - qca_core.h, 319
- qcaVersion
 - qca_core.h, 320
- QPipe
 - QCA::QPipe, 237
- qpipeline.h, 337
- Quiet
 - QCA::Logger, 191
- Random
 - QCA::Random, 240
- randomArray
 - QCA::Random, 241
- randomChar
 - QCA::Random, 240
- randomInt
 - QCA::Random, 241
- read
 - QCA::SASL, 254
 - QCA::SecureLayer, 267
 - QCA::SecureMessage, 276
 - QCA::TLS, 303
- readEnd
 - QCA::QPipe, 237
- readOutgoing
 - QCA::SASL, 254
 - QCA::SecureLayer, 267
 - QCA::TLS, 304
- readUnprocessed
 - QCA::SecureLayer, 267
 - QCA::TLS, 304
- readyReadOutgoing
 - QCA::SecureLayer, 268
- realmList
 - QCA::SASL, 252
- Reason
 - QCA::CRLEntry, 119
- reason

- QCA::CRLEntry, [121](#)
- recipientKeys
 - QCA::SecureMessage, [273](#)
- registerLogDevice
 - QCA::Logger, [192](#)
- reject
 - QCA::EventHandler, [139](#)
- RemoteUnavailable
 - QCA::SASL, [248](#)
- removeEntry
 - QCA::KeyStore, [176](#)
- RemoveFromCRL
 - QCA::CRLEntry, [120](#)
- reset
 - QCA::QPipe, [237](#)
 - QCA::SASL, [249](#)
 - QCA::SecureMessage, [273](#)
 - QCA::TLS, [297](#)
- resize
 - QCA::MemoryRegion, [197](#)
 - QCA::SecureArray, [262](#)
- revoked
 - QCA::CRL, [116](#)
- RSA
 - QCA::PKey, [216](#)
- RSAPrivateKey
 - QCA::RSAPrivateKey, [243](#)
- RSAPublicKey
 - QCA::RSAPublicKey, [245](#)
- SASL
 - QCA::SASL, [249](#)
- saveProviderConfig
 - QCA, [41](#)
- scanForPlugins
 - QCA, [40](#)
- Section
 - QCA::CertificateInfoType, [88](#)
 - QCA::ConstraintType, [111](#)
- section
 - QCA::CertificateInfoType, [88](#)
 - QCA::ConstraintType, [112](#)
- SecureArray
 - QCA::SecureArray, [259](#), [260](#)
- SecureLayer
 - QCA::SecureLayer, [266](#)
- SecureMessage
 - QCA::SecureMessage, [272](#)
- SecureMessageKey
 - QCA::SecureMessageKey, [281](#)
- SecureMessageKeyList
 - QCA, [30](#)
- SecureMessageSignature
 - QCA::SecureMessageSignature, [285](#)
- SecureMessageSignatureList
 - QCA, [30](#)
- SecureMessageSystem
 - QCA::SecureMessageSystem, [286](#)
- SecurityLevel
 - QCA, [35](#)
- serialNumber
 - QCA::Certificate, [69](#)
 - QCA::CertificateOptions, [93](#)
 - QCA::CRLEntry, [121](#)
- ServerAuth
 - QCA, [31](#)
- ServerSendMode
 - QCA::SASL, [249](#)
- serverStarted
 - QCA::SASL, [255](#)
- session
 - QCA::TLS, [301](#)
- set
 - QCA::MemoryRegion, [197](#)
 - QCA::PKey, [219](#)
 - QCA::SecureArray, [263](#)
- setAppName
 - QCA, [42](#)
- setAsCA
 - QCA::CertificateOptions, [95](#)
- setAsUser
 - QCA::CertificateOptions, [95](#)
- setAuthzid
 - QCA::SASL, [253](#)
- setBlockingEnabled
 - QCA::KeyGenerator, [164](#)
- setBundleSignerEnabled
 - QCA::SecureMessage, [274](#)
- setCertificate
 - QCA::TLS, [297](#)
- setCertificateChainAndKey
 - QCA::KeyBundle, [157](#)
- setChallenge
 - QCA::CertificateOptions, [93](#)
- setCompressionEnabled
 - QCA::TLS, [299](#)
- setConstraints
 - QCA::CertificateOptions, [94](#)
 - QCA::SASL, [249](#)
 - QCA::TLS, [298](#)
- setCRLLocations
 - QCA::CertificateOptions, [94](#)
- setExternalAuthId
 - QCA::SASL, [250](#)
- setExternalSSF
 - QCA::SASL, [250](#)
- setFileName
 - QCA::FileWatch, [142](#)

- setFormat
 - QCA::CertificateOptions, 91
 - QCA::SecureMessage, 274
- setGlobalRandomProvider
 - QCA, 41
- setInfo
 - QCA::CertificateOptions, 94
- setInfoOrdered
 - QCA::CertificateOptions, 94
- setIssuerList
 - QCA::TLS, 299
- setIssuerLocations
 - QCA::CertificateOptions, 94
- setLevel
 - QCA::Logger, 191
- setLineBreaksColumn
 - QCA::Base64, 53
- setLineBreaksEnabled
 - QCA::Base64, 53
- setLocalAddress
 - QCA::SASL, 250
- setName
 - QCA::KeyBundle, 157
- setOCSPLocations
 - QCA::CertificateOptions, 95
- setPacketMTU
 - QCA::TLS, 304
- setPassword
 - QCA::SASL, 253
- setPasswordData
 - QCA::Event, 136
- setPasswordKeyStore
 - QCA::Event, 136
- setPGPPublicKey
 - QCA::SecureMessageKey, 281
- setPGPSecretKey
 - QCA::SecureMessageKey, 282
- setPolicies
 - QCA::CertificateOptions, 94
- setPrivateKeys
 - QCA::CMS, 109
- setProperty
 - QCA, 41
- setProviderConfig
 - QCA, 41
- setProviderPriority
 - QCA, 39
- setRealm
 - QCA::SASL, 253
- setRecipient
 - QCA::SecureMessage, 274
- setRecipients
 - QCA::SecureMessage, 274
- setRemoteAddress
 - QCA::SASL, 250
- setSecure
 - QCA::MemoryRegion, 197
- setSerialNumber
 - QCA::CertificateOptions, 95
- setSession
 - QCA::TLS, 299
- setSigner
 - QCA::SecureMessage, 274
- setSigners
 - QCA::SecureMessage, 275
- setSMIMEAttributesEnabled
 - QCA::SecureMessage, 274
- setToken
 - QCA::Event, 136
- setTrustedCertificates
 - QCA::CMS, 109
 - QCA::TLS, 297
- setUntrustedCertificates
 - QCA::CMS, 109
- setup
 - QCA::Cipher, 107
 - QCA::MessageAuthenticationCode, 200
 - QCA::TextFilter, 291
- setUsername
 - QCA::SASL, 253
- setValidityPeriod
 - QCA::CertificateOptions, 95
- setX509CertificateChain
 - QCA::SecureMessageKey, 282
- setX509KeyBundle
 - QCA::SecureMessageKey, 282
- setX509PrivateKey
 - QCA::SecureMessageKey, 282
- Severity
 - QCA::Logger, 191
- signature
 - QCA::PrivateKey, 226
 - QCA::SecureMessage, 277
- SignatureAlgorithm
 - QCA, 33
- signatureAlgorithm
 - QCA::Certificate, 70
 - QCA::CertificateRequest, 99
 - QCA::CRL, 116
- SignatureFormat
 - QCA, 34
- SignatureUnknown
 - QCA, 34
- signer
 - QCA::SecureMessage, 278
- signerKeys
 - QCA::SecureMessage, 273
- signers

- QCA::SecureMessage, 278
- signMessage
 - QCA::PrivateKey, 226
- SignMode
 - QCA::SecureMessage, 271
- signRequest
 - QCA::CertificateAuthority, 75
- size
 - QCA::MemoryRegion, 196
 - QCA::SecureArray, 262
- SL_Baseline
 - QCA, 35
- SL_Export
 - QCA, 35
- SL_High
 - QCA, 35
- SL_Highest
 - QCA, 35
- SL_Integrity
 - QCA, 35
- SL_None
 - QCA, 35
- SmartCard
 - QCA::KeyStore, 173
- smimeAttributesEnabled
 - QCA::SecureMessage, 273
- Source
 - QCA::Event, 134
- source
 - QCA::Event, 135
- SPKAC
 - QCA, 30
- ssf
 - QCA::SASL, 252
- SSL_v2
 - QCA::TLS, 295
- SSL_v3
 - QCA::TLS, 295
- start
 - QCA::EventHandler, 139
 - QCA::KeyStoreManager, 188
- startAsynchronousMode
 - QCA::KeyStore, 174
- startClient
 - QCA::SASL, 250
 - QCA::TLS, 300
- startDecrypt
 - QCA::SecureMessage, 275
- startEncrypt
 - QCA::SecureMessage, 275
- startServer
 - QCA::SASL, 251
 - QCA::TLS, 300
- startSign
 - QCA::PrivateKey, 225
 - QCA::SecureMessage, 275
- startSignAndEncrypt
 - QCA::SecureMessage, 276
- startVerify
 - QCA::PublicKey, 233
 - QCA::SecureMessage, 276
- State
 - QCA, 31
- storeId
 - QCA::KeyStoreEntry, 181
- storeName
 - QCA::KeyStoreEntry, 181
- Stream
 - QCA::TLS, 295
- stringToArray
 - QCA::TextFilter, 292
- StylePassphrase
 - QCA::Event, 134
- StylePassword
 - QCA::Event, 134
- StylePIN
 - QCA::Event, 134
- subjectInfo
 - QCA::Certificate, 67
 - QCA::CertificateRequest, 98
- subjectInfoOrdered
 - QCA::Certificate, 68
 - QCA::CertificateRequest, 98
- subjectKeyId
 - QCA::Certificate, 70
- subjectPublicKey
 - QCA::Certificate, 69
 - QCA::CertificateRequest, 99
- submitPassword
 - QCA::EventHandler, 139
- success
 - QCA::SecureMessage, 277
- Superseded
 - QCA::CRLEntry, 120
- supportedCipherSuites
 - QCA::TLS, 297
- supportedFeatures
 - QCA, 38
- supportedGroupSets
 - QCA::DLGroup, 127
- supportedIOTypes
 - QCA::PKey, 217
- supportedPBESAlgorithms
 - QCA::PrivateKey, 226
- supportedTypes
 - QCA::PKey, 217
- SymmetricKey
 - QCA::SymmetricKey, 288, 289

- sync
 - QCA::KeyStoreManager, 188
- System
 - QCA::KeyStore, 173
- systemStore
 - QCA, 42
- takeContext
 - QCA::Algorithm, 51
- TextFilter
 - QCA::TextFilter, 291
- thisUpdate
 - QCA::CRL, 116
- time
 - QCA::CRLEntry, 121
- timestamp
 - QCA::SecureMessageSignature, 285
- TimeStamping
 - QCA, 32
- TLS
 - QCA::TLS, 296
- TLS_v1
 - QCA::TLS, 295
- toArray
 - QCA::BigInteger, 60
 - QCA::KeyBundle, 157
 - QCA::PGPKey, 213
- toByteArray
 - QCA::MemoryRegion, 195
 - QCA::SecureArray, 263
- toDER
 - QCA::Certificate, 71
 - QCA::CertificateRequest, 100
 - QCA::CRL, 117
 - QCA::PrivateKey, 226
 - QCA::PublicKey, 234
- toDH
 - QCA::PrivateKey, 224
 - QCA::PublicKey, 232
- toDHPrivateKey
 - QCA::PKey, 220
- toDHPublicKey
 - QCA::PKey, 220
- toDSA
 - QCA::PrivateKey, 224
 - QCA::PublicKey, 232
- toDSAPrivateKey
 - QCA::PKey, 220
- toDSAPublicKey
 - QCA::PKey, 220
- toFile
 - QCA::KeyBundle, 158
 - QCA::PGPKey, 213
- toFlatTextFile
 - QCA::CertificateCollection, 82
- Token
 - QCA::Event, 134
- TokenAsker
 - QCA::TokenAsker, 308
- tokenOkay
 - QCA::EventHandler, 139
- TooWeak
 - QCA::SASL, 248
- toPEM
 - QCA::Certificate, 71
 - QCA::CertificateRequest, 100
 - QCA::CRL, 117
 - QCA::PrivateKey, 227
 - QCA::PublicKey, 234
- toPEMFile
 - QCA::Certificate, 71
 - QCA::CertificateRequest, 100
 - QCA::CRL, 117
 - QCA::PrivateKey, 227
 - QCA::PublicKey, 234
- toPKCS7File
 - QCA::CertificateCollection, 82
- toPrivateKey
 - QCA::PKey, 219
- toPublicKey
 - QCA::PKey, 219
- toRSA
 - QCA::PrivateKey, 224
 - QCA::PublicKey, 232
- toRSAPrivateKey
 - QCA::PKey, 219
- toRSAPublicKey
 - QCA::PKey, 219
- toString
 - QCA::BigInteger, 60
 - QCA::CertificateInfoOrdered, 84
 - QCA::CertificateRequest, 101
 - QCA::KeyStoreEntry, 181
 - QCA::PGPKey, 213
- trustedCertificates
 - QCA::CMS, 109
 - QCA::TLS, 297
- Type
 - QCA::Event, 134
 - QCA::KeyStore, 173
 - QCA::KeyStoreEntry, 179
 - QCA::PKey, 216
 - QCA::SecureMessage, 271
 - QCA::SecureMessageKey, 280
- type
 - QCA::Algorithm, 50
 - QCA::CertificateInfoPair, 86
 - QCA::Cipher, 105

- QCA::Event, 135
- QCA::Hash, 146
- QCA::KeyStore, 174
- QCA::KeyStoreEntry, 180
- QCA::KeyStoreInfo, 186
- QCA::MessageAuthenticationCode, 199
- QCA::PKey, 218
- QCA::SecureMessage, 272
- QCA::SecureMessageKey, 281
- unavailable
 - QCA::KeyStore, 176
 - QCA::KeyStoreEntryWatcher, 184
- unloadAllPlugins
 - QCA, 40
- unregisterLogDevice
 - QCA::Logger, 192
- Unspecified
 - QCA::CRLEntry, 120
- untrustedCertificates
 - QCA::CMS, 109
- update
 - QCA::Base64, 53
 - QCA::BufferedComputation, 63
 - QCA::Cipher, 106
 - QCA::Filter, 143
 - QCA::Hash, 147
 - QCA::Hex, 151
 - QCA::MessageAuthenticationCode, 200
 - QCA::PrivateKey, 225
 - QCA::PublicKey, 233
 - QCA::SecureMessage, 276
- updateCRL
 - QCA::CertificateAuthority, 76
- URI
 - QCA, 31
- UsageAny
 - QCA, 32
- UsageCodeSigning
 - QCA, 32
- UsageCRLSigning
 - QCA, 32
- UsageEmailProtection
 - QCA, 32
- UsageMode
 - QCA, 32
- UsageTimeStamping
 - QCA, 32
- UsageTLSClient
 - QCA, 32
- UsageTLSServer
 - QCA, 32
- User
 - QCA::KeyStore, 173
- userIds
 - QCA::PGPKey, 212
- Valid
 - QCA::SecureMessageSignature, 284
 - QCA::TLS, 296
- validate
 - QCA::Certificate, 70
 - QCA::CertificateChain, 78
- ValidateFlags
 - QCA, 33
- Validity
 - QCA, 32
- ValidityGood
 - QCA, 32
- validKeyLength
 - QCA::Cipher, 106
 - QCA::MessageAuthenticationCode, 199
- validSignature
 - QCA::PublicKey, 233
- value
 - QCA::CertificateInfoPair, 86
- verifyMessage
 - QCA::PublicKey, 234
- verifySuccess
 - QCA::SecureMessage, 278
- Version
 - QCA::TLS, 295
- version
 - QCA::TLS, 301
- waitForBusyFinished
 - QCA::KeyStoreManager, 188
- waitForFinished
 - QCA::SecureMessage, 277
- waitForResponse
 - QCA::PasswordAsker, 204
 - QCA::TokenAsker, 308
- Warning
 - QCA::Logger, 191
- wasSigned
 - QCA::SecureMessage, 278
- withAlgorithm
 - QCA::KeyDerivationFunction, 162
- withAlgorithms
 - QCA::Cipher, 107
- write
 - QCA::SASL, 254
 - QCA::SecureLayer, 267
 - QCA::TLS, 303
- writeEnd
 - QCA::QPipe, 238
- writeEntry
 - QCA::KeyStore, 175, 176

writeIncoming

QCA::SASL, [254](#)

QCA::SecureLayer, [267](#)

QCA::TLS, [303](#)

x

QCA::DHPrivateKey, [123](#)

QCA::DSAPrivateKey, [130](#)

X509

QCA::SecureMessageKey, [281](#)

x509CertificateChain

QCA::SecureMessageKey, [282](#)

x509PrivateKey

QCA::SecureMessageKey, [282](#)

XMPP

QCA, [31](#)

y

QCA::DHPrivateKey, [123](#)

QCA::DHPublicKey, [125](#)

QCA::DSAPrivateKey, [130](#)

QCA::DSAPublicKey, [132](#)