PictureCrypt

1.3.5

Generated by Doxygen 1.8.6

Tue Jan 1 2019 19:38:08

Contents

Chapter 1

PictureCrypt

Project made using QT Creator in C++

1.1 About

A simple steganography project which hides data in images This project is built using MVC pattern and features GUI. `Qt` and `QAESEncryption` by `bricke` were used.

1.2 Download

Get the binary files at `latest release page` Or download latest **UNSTABLE** binary file for linux `here`

1.3 Realisation

To create the encrypted image, you need to select any file for encryption, then using EncryptDialog you select the image to store the data. Then output image is generated.

Attention

> Output image format available is .PNG, because .jpg isn't lossless, so the pixels containing data would be seriously simplified and the data damaged. .BMP isn't used, because noone really uses it and .PNG is just compressed .BMP (more or less)

Note

> JPHS support is under development :D

1.4 How can someone use it?

Well... Anyone who wants to securely commuicate. For example your boss watches your inbox, so you do the work and don't chat with your friends about the bar, they've just visited. Using this app you can send them a photo of your desk, saying it's my new working space, but inside the image there is secret message saying "Wanna get another beer tonight? xD". Boss sees this image, but doesn't spot anyhing. Great example...

1.5 Structure of the project.

Project is done via MVC Pattern. View and Model layers are totally isolated and run on different threads.

Code from controller.cpp

```cpp
view = new ViewPC();
model = new ModelPC(version);
QThread * modelThread = new QThread();
model->moveToThread(modelThread);
modelThread->start();
```

So when Model is hard-working, View layer is just fine.

Layers also have a ton of functions, so here is a scheme, that I was doing for about 10 hours, which demonstrates the most important functions and classes in the project. And everything is clickable here, so try it out!
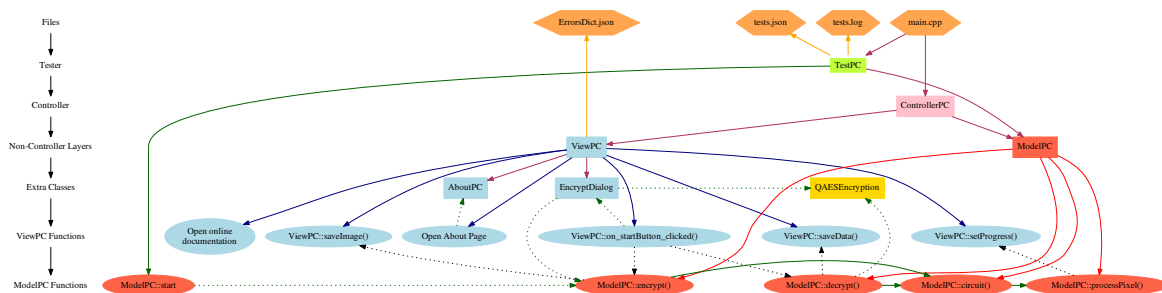
Well... I think you didn't quite understand what is happening here... So hop into my "User-friendly" Documentation!

See source on https://github.com/waleko/PictureCrypt

Note

> QAESEncryption class done by Bricke

1.6 External use

ModelPC class can be used externally (without UI)

Note

> TestPC class was introduced recently, its use is adviced.

```cpp
#include <modelpc.h>
#include <testpc.h>
#include <QByteArray>
#include <QImage>

#include <QDebug> // Just for demonstration use

...

if(TestPC::Test())
    return;
ModelPC * model = new ModelPC();

// Embedding
QImage * resultImage = model->start(QByteArray data, // Data to be embedded
                                    QImage *image, // Image for embedding
                                    int mode = 0, // Mode of embedding
                                    QString key = "", // Key for extra-encryption (if empty, key will be
    generated automatically)
                                    int bitsUsed = 8, // Bits per Byte used (better explaination
    ModelPC::bitsUsed)
                                    QString *error = nullptr); // Error output, if everything is ok, error
    will be "ok"
if(*error != "ok")
    return;
// Note *error is just a code of error (like "muchdata", dictionary of error codes is also available on
```

```
            github.

// De-embedding
QByteArray output = model->decrypt(QImage * image, // Image with hidden data
                                   QString *_error = nullptr); // Error output
if(data == output)
   qDebug() << "Great success!";
else
   qDebug() << "Fiasco :(";
```

See Also

ModelPC, ModelPC::ModelPC, ModelPC::saveData, ModelPC::saveImage, ModelPC::alertView, ModelPC-::setProgress

1.7 JPHS use

The newer versions of the app have jphs support, but they don't have jphs built in as it is provided under GNU General Public License v3.0, is "for test purposes only" and is illegal in some countries, so...

Attention

We support JPHS, but we don't use any responsibility for it, we never used or downloaded it, we just used .exe output in the web, and it somehow works by chance. All responsibility for using jphs is on you, that is why we use made only optionally. That means that to use jphs with our app you will have to download the jphs yourself and specify the jphs directory. However we provide link to the site where you can download the supported version of the jphs: http://linux01.gwdg.de/~alatham/stego.html As it's not our site publishing the dangerous zip file, we just put link to that site (Google does that too, so what? Sue Google?), This text is subject to United Nations' Universal Declaration of Human Rights, (see Article 19 http-://www.un.org/en/universal-declaration-human-rights):

> Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers.

And I typed this link randomly, and I'm scared...

1.8 License

This software is provided under the UNLICENSE

1.9 Contact us

Visit my site: https://www.alexkovrigin.me

Email me at a.kovrigin0@gmail.com

Author

Alex Kovrigin (waleko)

Copyright

Alex Kovrigin 2018

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 ErrorsDictSetup Namespace Reference

Variables

- string filename = 'ErrorsDict.json'
- tuple raw = open(filename, 'r')
- tuple data = json.load(raw)
- tuple input_data = input()

6.1.1 Variable Documentation

6.1.1.1 tuple ErrorsDictSetup.data = json.load(raw)

Definition at line 6 of file ErrorsDictSetup.py.

6.1.1.2 string ErrorsDictSetup.filename = 'ErrorsDict.json'

Definition at line 2 of file ErrorsDictSetup.py.

6.1.1.3 tuple ErrorsDictSetup.input_data = input()

Definition at line 14 of file ErrorsDictSetup.py.

6.1.1.4 tuple ErrorsDictSetup.raw = open(filename, 'r')

Definition at line 4 of file ErrorsDictSetup.py.

6.2 tests-setup Namespace Reference

Variables

- string filename = 'tests.json'
- tuple raw = open(filename, 'r')
- tuple js = json.load(raw)
- tuple input_data = input()
- list arr = []

- dictionary [obj] = {'data':data, 'image':image,'expectation':expect,'mode':int(mode),'key':key,'bitsUsed':int(bits-Used)}

6.2.1 Variable Documentation

6.2.1.1 list tests-setup.arr = [ ]

Definition at line 16 of file tests-setup.py.

6.2.1.2 string tests-setup.filename = 'tests.json'

Definition at line 2 of file tests-setup.py.

6.2.1.3 tuple tests-setup.input_data = input()

Definition at line 14 of file tests-setup.py.

6.2.1.4 tuple tests-setup.js = json.load(**raw**)

Definition at line 6 of file tests-setup.py.

6.2.1.5 dictionary tests-setup.obj = {'data':data, 'image':image,'expectation':expect,'mode':int(mode),'key':key,'bitsUsed':int(bits-Used)}

Definition at line 20 of file tests-setup.py.

6.2.1.6 tuple tests-setup.raw = open(**filename**, 'r')

Definition at line 4 of file tests-setup.py.

6.3 Ui Namespace Reference

Chapter 7

Class Documentation

7.1 AboutPC Class Reference

The AboutPC class The About Page dialog.

```
#include <aboutpc.h>
```

Inheritance diagram for AboutPC:

```
QDialog
   ↑
AboutPC
```

Collaboration diagram for AboutPC:

```
QDialog
   ↑
AboutPC
```

Public Member Functions

- AboutPC (QWidget ∗parent=0)
- ∼AboutPC ()
- void setVersion (QString version)

    AboutPC::setVersion Function to set the version display.

7.1.1 Detailed Description

The AboutPC class The About Page dialog.

Definition at line 12 of file aboutpc.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 AboutPC::AboutPC (QWidget ∗ *parent =* 0) `[explicit]`

Definition at line 4 of file aboutpc.cpp.

7.1.2.2 AboutPC::∼AboutPC ( )

Definition at line 11 of file aboutpc.cpp.

7.1.3 Member Function Documentation

7.1.3.1 void AboutPC::setVersion (QString *version*)

AboutPC::setVersion Function to set the version display.

Parameters

version	Version as QString

Definition at line 19 of file aboutpc.cpp.

Here is the caller graph for this function:
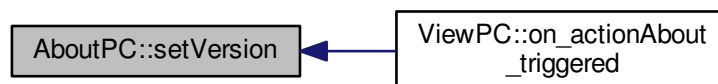
The documentation for this class was generated from the following files:

- aboutpc.h
- aboutpc.cpp

7.2 ControllerPC Class Reference

The ControllerPC class Controller class, which controls View and Model layers.

```
#include <controllerpc.h>
```

Inheritance diagram for ControllerPC:

Collaboration diagram for ControllerPC:
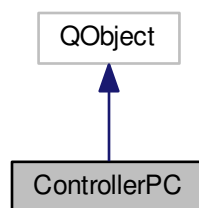
Public Slots

- void abortCircuit ()

  ControllerPC::abortCircuit Slot to be called when ProgressDialog in ViewPC is closed. It flags ModelPC to stop.
- void runTests ()

  ControllerPC::runTests Runs tests.
- void setJPHSDir (QString dir)

  ControllerPC::setJPHSDir Sets JPHS default dir.

Public Member Functions

- ControllerPC ()

  ControllerPC::ControllerPC Constructor of controller Constructor runs auto-test for ModelPC, creates Model Class (ModelPC) and View Class (ViewPC). All signals and slots are connected here.

Public Attributes

- long int version

    version Version of the app

- QString versionString

    versionString Version of the app as QString.

7.2.1 Detailed Description

The ControllerPC class Controller class, which controls View and Model layers.

See Also

    ViewPC, ModelPC

Definition at line 21 of file controllerpc.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 ControllerPC::ControllerPC ( )

ControllerPC::ControllerPC Constructor of controller Constructor runs auto-test for ModelPC, creates Model Class (ModelPC) and View Class (ViewPC). All signals and slots are connected here.

Controller class

Note

    Version of the app is specified here.

Definition at line 9 of file controllerpc.cpp.

Here is the call graph for this function:
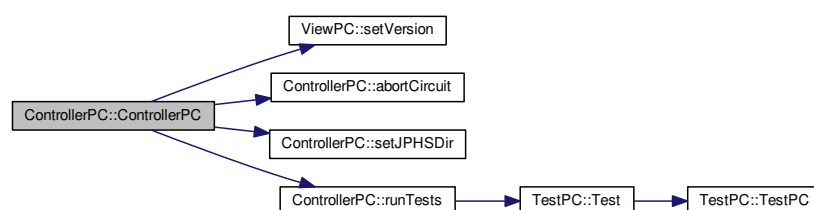
7.2.3 Member Function Documentation

7.2.3.1 void ControllerPC::abortCircuit ( ) `[slot]`

ControllerPC::abortCircuit Slot to be called when ProgressDialog in ViewPC is closed. It flags ModelPC to stop.

Definition at line 37 of file controllerpc.cpp.

Here is the caller graph for this function:

7.2.3.2 void ControllerPC::runTests ( ) `[slot]`

ControllerPC::runTests Runs tests.

Definition at line 44 of file controllerpc.cpp.

Here is the call graph for this function:
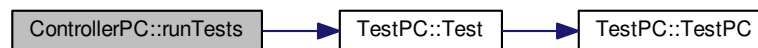
Here is the caller graph for this function:

7.2.3.3 void ControllerPC::setJPHSDir (QString _dir_) `[slot]`

ControllerPC::setJPHSDir Sets JPHS default dir.

Parameters

dir	Directory

Definition at line 56 of file controllerpc.cpp.

Here is the caller graph for this function:

7.2.4 Member Data Documentation

7.2.4.1 long int ControllerPC::version

version Version of the app

Definition at line 29 of file controllerpc.h.

7.2.4.2 QString ControllerPC::versionString

versionString Version of the app as QString.

Definition at line 33 of file controllerpc.h.

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

- controllerpc.h
- controllerpc.cpp

7.3 EncryptDialog Class Reference

The EncryptDialog class Class to get the image and key to store secret info.

```
#include <encryptdialog.h>
```

Inheritance diagram for EncryptDialog:
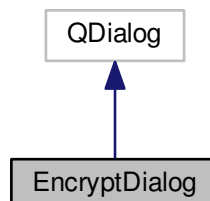
Collaboration diagram for EncryptDialog:

Public Slots

- void on_fileButton_clicked ()

  EncryptDialog::on_fileButton_clicked Slot to select the image.
- void on_buttonBox_accepted ()

  EncryptDialog::on_buttonBox_accepted Slot to start the encryption. Successful closing of the app.
- void on_buttonBox_rejected ()

  EncryptDialog::on_buttonBox_rejected Slot to reject the encryption.
- void on_bitsSlider_valueChanged (int value)

  EncryptDialog::on_bitsSlider_valueChanged Slot if value of the bits slider is changed.

Public Member Functions

- EncryptDialog (QByteArray _data, QWidget ∗parent=0)

  EncryptDialog::EncryptDialog Constructor of the class. Input data is saved here and some variables are set here.
- ∼EncryptDialog ()
- QByteArray zip ()

  EncryptDialog::zip Zipping algorithm It copresses the data and then compresses it using qCompress()

Public Attributes

- QByteArray data

  data Input data
- bool success

  success Flag, if image was successfully selected and data was encrypted.
- QByteArray compr_data

  compr_data Compressed data, aka Output data.
- QString inputFileName

  inputFileName Filename of the image.
- long long int size

  size Size of the image in square pixels
- QString key

  key Key to be used for encryption in EncrytDialog::zip
- bool goodPercentage

  goodPercentage Flag if area of the used data via encryption is less than 70% of the area of the image.

- int val

  val Value of the slider
- int bitsUsed

  bitsUsed Bits used per byte of pixel.
- QImage image

  image Inputted image

7.3.1 Detailed Description

The EncryptDialog class Class to get the image and key to store secret info.

Note

> Not the most important and well written class.

See Also

> ViewPC

Definition at line 21 of file encryptdialog.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 EncryptDialog::EncryptDialog (QByteArray _data, QWidget * parent = 0) `[explicit]`

EncryptDialog::EncryptDialog Constructor of the class. Input data is saved here and some variables are set here.

Parameters

_data	Input data.
parent	Parent (not in use)

Definition at line 9 of file encryptdialog.cpp.

Here is the call graph for this function:
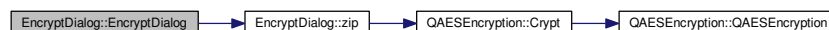
7.3.2.2 EncryptDialog::∼EncryptDialog ( )

Definition at line 26 of file encryptdialog.cpp.

7.3.3 Member Function Documentation

7.3.3.1 void EncryptDialog::on_bitsSlider_valueChanged (int value) `[slot]`

EncryptDialog::on_bitsSlider_valueChanged Slot if value of the bits slider is changed.

Parameters

value	Well, value

Definition at line 107 of file encryptdialog.cpp.

7.3.3.2 void EncryptDialog::on_buttonBox_accepted ( ) `[slot]`

EncryptDialog::on_buttonBox_accepted Slot to start the encryption. Successful closing of the app.

Definition at line 82 of file encryptdialog.cpp.

Here is the call graph for this function:

7.3.3.3 void EncryptDialog::on_buttonBox_rejected ( ) `[slot]`

EncryptDialog::on_buttonBox_rejected Slot to reject the encryption.

Definition at line 98 of file encryptdialog.cpp.

7.3.3.4 void EncryptDialog::on_fileButton_clicked ( ) `[slot]`

EncryptDialog::on_fileButton_clicked Slot to select the image.

Definition at line 57 of file encryptdialog.cpp.

7.3.3.5 QByteArray EncryptDialog::zip ( )

EncryptDialog::zip Zipping algorithm It copresses the data and then compresses it using qCompress()

Returns

Returns Compressed data.

See Also

ModelPC::unzip

Definition at line 46 of file encryptdialog.cpp.

Here is the call graph for this function:
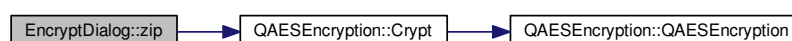
Here is the caller graph for this function:

7.3.4 Member Data Documentation

7.3.4.1 int EncryptDialog::bitsUsed

bitsUsed Bits used per byte of pixel.

See Also

> ModelPC::circuit

Definition at line 75 of file encryptdialog.h.

7.3.4.2 QByteArray EncryptDialog::compr_data

compr_data Compressed data, aka Output data.

Definition at line 50 of file encryptdialog.h.

7.3.4.3 QByteArray EncryptDialog::data

data Input data

Definition at line 42 of file encryptdialog.h.

7.3.4.4 bool EncryptDialog::goodPercentage

goodPercentage Flag if area of the used data via encryption is less than 70% of the area of the image.

Definition at line 66 of file encryptdialog.h.

7.3.4.5 QImage EncryptDialog::image

image Inputted image

Definition at line 79 of file encryptdialog.h.

7.3.4.6 QString EncryptDialog::inputFileName

inputFileName Filename of the image.

Definition at line 54 of file encryptdialog.h.

7.3.4.7 QString EncryptDialog::key

key Key to be used for encryption in EncrytDialog::zip

Definition at line 62 of file encryptdialog.h.

7.3.4.8 long long int EncryptDialog::size

size Size of the image in square pixels

Definition at line 58 of file encryptdialog.h.

7.3.4.9 bool EncryptDialog::success

success Flag, if image was successfully selected and data was encrypted.

Definition at line 46 of file encryptdialog.h.

7.3.4.10 int EncryptDialog::val

val Value of the slider

Definition at line 70 of file encryptdialog.h.

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

- encryptdialog.h
- encryptdialog.cpp

7.4 ModelPC Class Reference

The ModelPC class Model Layer of the app. Main class that does the work of PictureCrypt logic Controled by ControllerPC.

```
#include <modelpc.h>
```

Inheritance diagram for ModelPC:
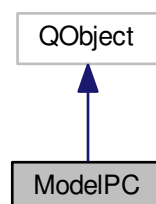
Collaboration diagram for ModelPC:

QObject
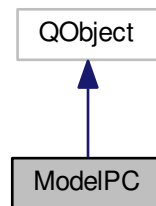
ModelPC

Public Types

- enum CryptMode { NotDefined, v1_3, v1_4, jphs_mode }

Public Slots

- QImage ∗ encrypt (QByteArray data, QImage ∗image, int _mode, QString key="", int _bitsUsed=8, QString ∗_error=nullptr)

    ModelPC::encrypt Slot to zip and inject data and provide it with some extra stuff After completion start standard ModelPC::inject Isn't used in PictureCrypt, but used can be used in other - custom projects.
- QImage ∗ inject (QByteArray encr_data, QImage ∗image, int _mode, int _bitsUsed=8, QString ∗_-error=nullptr)

    ModelPC::inject Slot to be called when encrypt mode in ViewPC is selected and started.
- QByteArray decrypt (QImage ∗image, QString key, int _mode=0, QString ∗_error=nullptr)

    ModelPC::decrypt Slot to be called when decrypt mode in ViewPC is selected and started.
- void fail (QString message)

    ModelPC::fail Slot to stop execution of cryption.
- void alert (QString message, bool isWarning=false)

    ModelPC::alert Function emits signal ModelPC::alertView and calls ViewPC::alert.

Signals

- void alertView (QString messageCode, bool isWarning)

    alertView Signal to be called to create MessageBox.
- void saveData (QByteArray data)

    saveData Signal to be called to save data from ModelPC::decrypt.
- void saveImage (QImage ∗image)

    saveImage Signal to be called to save image from ModelPC::encrypt.
- void setProgress (int val)

    setProgress Signal to be called to set progress of ProgressDialog.

Public Member Functions

- ModelPC ()

    ModelPC::ModelPC Constructor Unit tests are run here.
- QByteArray unzip (QByteArray data, QByteArray key)

    ModelPC::unzip Unzip data from ModelPC::decrypt. Just mirrored EncryptDialog::zip.

Static Public Member Functions

- static QImage ∗ [Encrypt](#) (QByteArray data, QImage ∗image, int _mode, QString key="", int _bitsUsed=8, QString ∗_error=nullptr)
- static QImage ∗ [Inject](#) (QByteArray encr_data, QImage ∗image, int _mode, int _bitsUsed=8, QString ∗_-error=nullptr)
- static QByteArray [Decrypt](#) (QImage ∗image, QString key, int _mode=0, QString ∗_error=nullptr)

Public Attributes

- bool [success](#)

    success Flag that true by default, but in case of error or cancelling of ProgressDialog it turns to false, which stops execution of [ModelPC::circuit](#)
- long [version](#)

    version Version of the class
- QString [versionString](#)

    versionString Version as string
- QString [defaultJPHSDir](#)

    defaultJPHSDir Default JPHS directory

Protected Member Functions

- void [circuit](#) (QImage ∗image, QByteArray ∗data, long long int countBytes)

    [ModelPC::circuit](#) The brain of the app. Via special circuit stores data in image.
- void [jphs](#) (QImage ∗image, QByteArray ∗data)

    [ModelPC::jphs](#) JPHS function to use jphide and jpseek (currently under development)
- void [processPixel](#) (QPoint pos, QVector< QPoint > ∗were, bool isEncrypt)

    [ModelPC::processPixel](#) Processes every pixel. Reads its contains or writes data.
- void [encryptv1_4](#) (QImage ∗image, QByteArray data, QString key)

    [ModelPC::encryptv1_4](#) Encrypts and injects data to image used in v1.4+.
- QByteArray [decryptv1_3](#) (QImage ∗image, QString key)

    [ModelPC::decryptv1_3](#) Decrytps data from image in v1.3.
- QByteArray [decryptv1_4](#) (QImage ∗image, QString key)

    [ModelPC::decryptv1_4](#) Decrypts data from image in v1.4+.
- QByteArray [zip](#) (QByteArray data, QByteArray key)

    [ModelPC::zip](#) Zip function, copy of [EncryptDialog::zip](#) Used for [ModelPC](#) in custom projects, other than PictureCrypt.

Protected Attributes

- QString ∗ [error](#)

    error Current error

7.4.1 Detailed Description

The [ModelPC](#) class Model Layer of the app. Main class that does the work of PictureCrypt logic Controled by [ControllerPC](#).

See Also

   [ViewPC](#), [ControllerPC](#)

Author

Alex Kovrigin (waleko)

Definition at line 30 of file modelpc.h.

7.4.2   Member Enumeration Documentation

7.4.2.1   enum **ModelPC::CryptMode**

Enumerator

> ***NotDefined***
>
> ***v1_3***
>
> ***v1_4***
>
> ***jphs_mode***

Definition at line 35 of file modelpc.h.

7.4.3   Constructor & Destructor Documentation

7.4.3.1   ModelPC::ModelPC ( )

ModelPC::ModelPC Constructor Unit tests are run here.

See Also

ControllerPC, ViewPC

Definition at line 9 of file modelpc.cpp.

Here is the caller graph for this function:
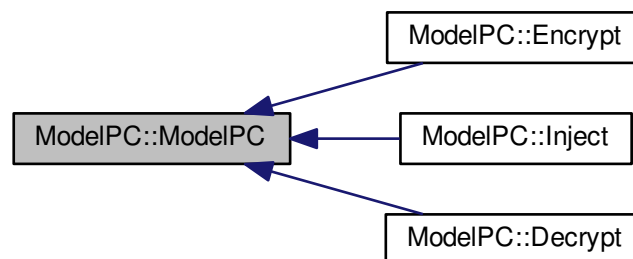
7.4.4   Member Function Documentation

7.4.4.1   void ModelPC::alert (QString *message,* bool *isWarning =* `false`) `[slot]`

ModelPC::alert Function emits signal ModelPC::alertView and calls ViewPC::alert.

Parameters

message	Message to be transmitted.
isWarning	Flag if message is critical.

See Also

> ViewPC::alert

Definition at line 703 of file modelpc.cpp.

Here is the caller graph for this function:
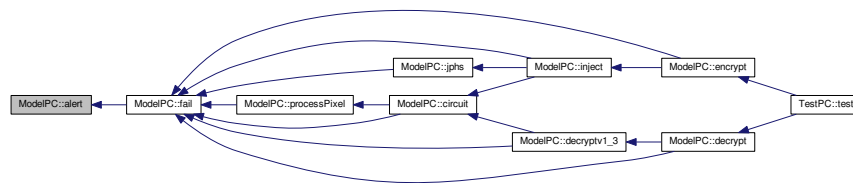
7.4.4.2 void ModelPC::alertView (QString *messageCode,* bool *isWarning*) `[signal]`

alertView Signal to be called to create MessageBox.

Parameters

messageCode	Message Code to be shown.
isWarning	Flag if message is critical.

See Also

> ModelPC::alert, ViewPC::alert

Here is the caller graph for this function:
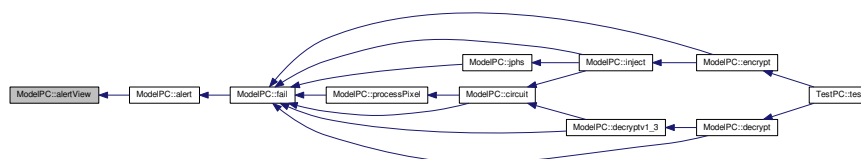
7.4.4.3 void ModelPC::circuit (QImage ∗ *image,* QByteArray ∗ *data,* long long int *countBytes*) `[protected]`

ModelPC::circuit The brain of the app. Via special circuit stores data in image.

The circuit itself can be found in documentation or in commentaries in source.

Parameters

image	Image to be processed.
data	Data to be processed.
countBytes	Number of bytes to be read or written.

See Also

> ModelPC::processPixel

Definition at line 308 of file modelpc.cpp.

Here is the call graph for this function:
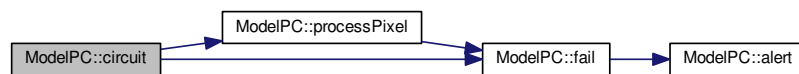
Here is the caller graph for this function:

7.4.4.4 QByteArray ModelPC::Decrypt (QImage ∗ *image,* QString *key,* int *_mode =* 0*,* QString ∗ *_error =* nullptr) [static]

Definition at line 34 of file modelpc.cpp.

Here is the call graph for this function:

7.4.4.5 QByteArray ModelPC::decrypt (QImage ∗ *image,* QString *key,* int *_mode =* 0*,* QString ∗ *_error =* nullptr) [slot]

ModelPC::decrypt Slot to be called when decrypt mode in ViewPC is selected and started.

Parameters

image	Image to be decrypted.
key	Keyphrase with which the data is injected
_mode	Mode for decryption
_error	Error output

Returns

Returns decrypted data

See Also

ViewPC::on_startButton_clicked, ModelPC::inject, ModelPC::circuit

Definition at line 195 of file modelpc.cpp.

Here is the call graph for this function:
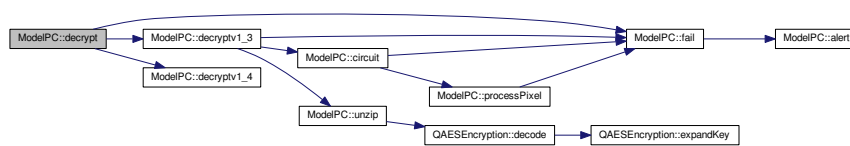
Here is the caller graph for this function:

7.4.4.6 QByteArray ModelPC::decryptv1_3 (QImage * *image,* QString *key*) `[protected]`

ModelPC::decryptv1_3 Decrytps data from image in v1.3.

Parameters

image	Image with data
key	Key

Returns

Returns obtained data

Definition at line 535 of file modelpc.cpp.

Here is the call graph for this function:
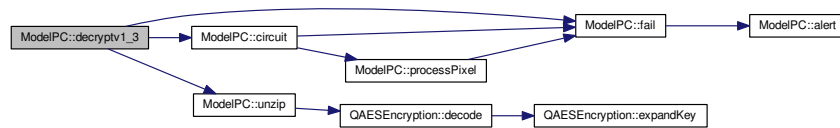
Here is the caller graph for this function:
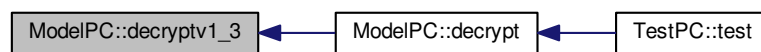
7.4.4.7 QByteArray ModelPC::decryptv1_4 (QImage ∗ _image,_ QString _key_) `[protected]`

ModelPC::decryptv1_4 Decrypts data from image in v1.4+.

Parameters

image	Image with data
key	Key

Returns

Returns obtained data

Definition at line 524 of file modelpc.cpp.

Here is the caller graph for this function:
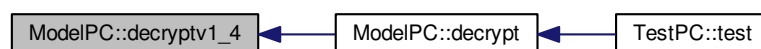
7.4.4.8 QImage ∗ ModelPC::Encrypt (QByteArray *data,* QImage ∗ *image,* int *_mode,* QString *key* = " ", int *_bitsUsed* = 8,
QString ∗ *_error* = nullptr) `[static]`

Definition at line 24 of file modelpc.cpp.

Here is the call graph for this function:

7.4.4.9 QImage ∗ ModelPC::encrypt (QByteArray *data,* QImage ∗ *image,* int *_mode,* QString *key = " ",* int *_bitsUsed =* 8*,* QString ∗ *_error =* nullptr) [slot]

ModelPC::encrypt Slot to zip and inject data and provide it with some extra stuff After completion start standard ModelPC::inject Isn't used in PictureCrypt, but used can be used in other - custom projects.

Parameters

data	Data for embedding
image	Image for embedding
mode	Mode for embedding
key	Key for extra encryption
_bitsUsed	Bits per byte (see ModelPC::bitsUsed)
_error	Error output

Returns

Returns image with embedded data

See Also

ModelPC::inject

Definition at line 51 of file modelpc.cpp.

Here is the call graph for this function:
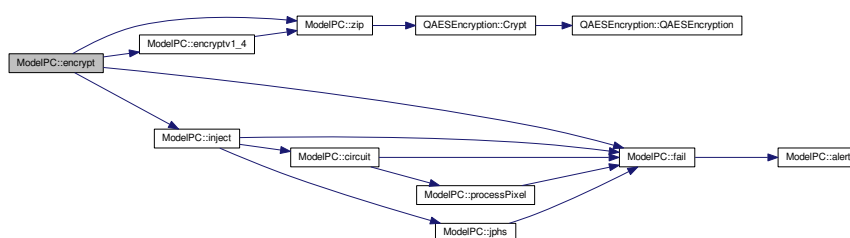
Here is the caller graph for this function:

7.4.4.10   void ModelPC::encryptv1_4 (QImage ∗ *image,* QByteArray *data,* QString *key*) `[protected]`

ModelPC::encryptv1_4 Encrypts and injects data to image used in v1.4+.

Parameters

image	Image for injecting
data	Data for embedding

Definition at line 509 of file modelpc.cpp.

Here is the call graph for this function:
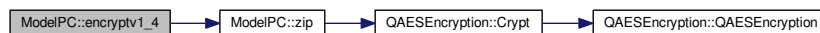
Here is the caller graph for this function:
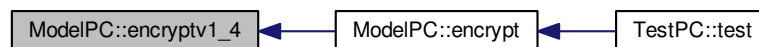
7.4.4.11   void ModelPC::fail (QString *message*) `[slot]`

ModelPC::fail Slot to stop execution of cryption.

Parameters

message	Message for user

Definition at line 235 of file modelpc.cpp.

Here is the call graph for this function:

```
ModelPC::fail ──────▶ ModelPC::alert
```

Here is the caller graph for this function:

```
ModelPC::jphs ◀──── ModelPC::inject ◀──── ModelPC::encrypt
ModelPC::fail ◀──── ModelPC::processPixel ◀──── ModelPC::circuit          TestPC::test
              ◀──── ModelPC::decryptv1_3 ◀──── ModelPC::decrypt
```
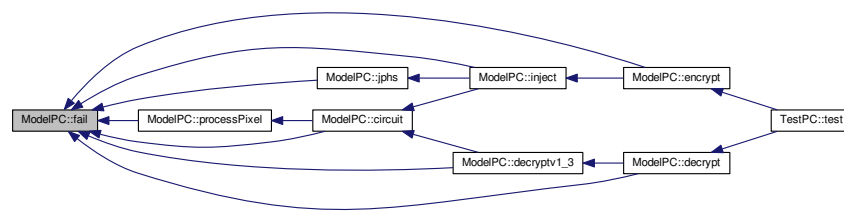
7.4.4.12   QImage ∗ ModelPC::Inject (QByteArray *encr_data,* QImage ∗ *image,* int *_mode,* int *_bitsUsed =* 8*,* QString ∗ *_error =* `nullptr` **) `[static]`

Definition at line 29 of file modelpc.cpp.

Here is the call graph for this function:

```
ModelPC::Inject ──────▶ ModelPC::ModelPC
```

7.4.4.13   QImage ∗ ModelPC::inject (QByteArray *encr_data,* QImage ∗ *image,* int *_mode,* int *_bitsUsed =* 8*,* QString ∗ *_error =* `nullptr` **) `[slot]`

ModelPC::inject Slot to be called when encrypt mode in ViewPC is selected and started.

Parameters

encr_data	Data to be inserted to an image.
image	Image to be inserted in.

mode	Mode of encryption
_bitsUsed	Bits per byte used
_error	Error output

Returns

Returns image with embedded data.

See Also

ViewPC::on_startButton_clicked, ModelPC::decrypt, ModelPC::circuit, ModelPC::start

Definition at line 130 of file modelpc.cpp.

Here is the call graph for this function:
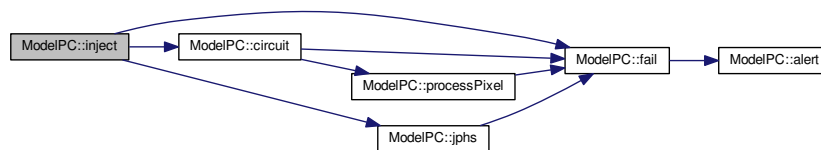
Here is the caller graph for this function:
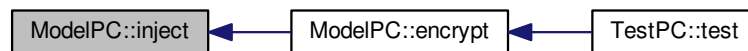
7.4.4.14   void ModelPC::jphs (QImage ∗ *image,* QByteArray ∗ *data*) [protected]

ModelPC::jphs JPHS function to use jphide and jpseek (currently under development)

Parameters

image	Image for embedding
data	Data

Definition at line 247 of file modelpc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.4.4.15   void ModelPC::processPixel (QPoint *pos,* QVector< QPoint > ∗ *were,* bool *isEncrypt*) `[protected]`

ModelPC::processPixel Processes every pixel. Reads its contains or writes data.

Parameters

pos	Position of pixel
were	Vector array containing pixels, that were already processed.
isEncrypt	Mode of operation. If true encryption operations will continue, else the decryption ones.

Definition at line 449 of file modelpc.cpp.

Here is the call graph for this function:
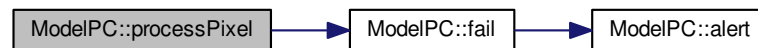
Here is the caller graph for this function:

7.4.4.16   void ModelPC::saveData (QByteArray *data*) `[signal]`

saveData Signal to be called to save data from ModelPC::decrypt.

Parameters

data	Data to be saved.

Here is the caller graph for this function:

7.4.4.17 void ModelPC::saveImage (QImage ∗ *image*) `[signal]`

saveImage Signal to be called to save image from ModelPC::encrypt.

Parameters

image	Image to be saved.

Here is the caller graph for this function:

7.4.4.18 void ModelPC::setProgress (int *val*) `[signal]`

setProgress Signal to be called to set progress of ProgressDialog.

Parameters

val	Value to be set.

See Also

ViewPC::setProgress

Here is the caller graph for this function:
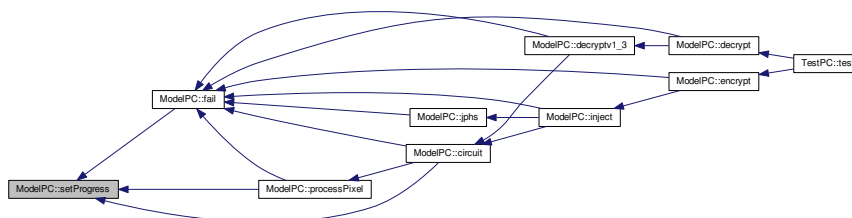
7.4.4.19 QByteArray ModelPC::unzip (QByteArray *data,* QByteArray *key*)

ModelPC::unzip Unzip data from ModelPC::decrypt. Just mirrored EncryptDialog::zip.

Parameters

data	Data to be decrypted.
key	Key to decrypt the data.

Returns
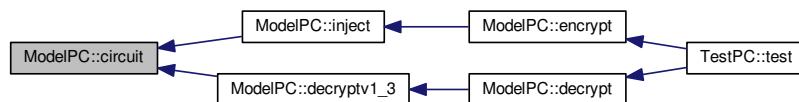
Returns data

See Also

EncryptDialog::zip, ModelPC::decrypt, ModelPC::zip

Definition at line 642 of file modelpc.cpp.

Here is the call graph for this function:
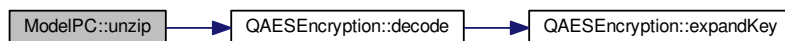
Here is the caller graph for this function:
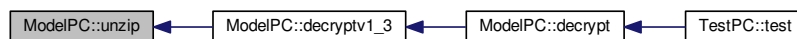
7.4.4.20 QByteArray ModelPC::zip (QByteArray *data,* **QByteArray** *key* **)** `[protected]`

ModelPC::zip Zip function, copy of EncryptDialog::zip Used for ModelPC in custom projects, other than PictureCrypt.

Parameters

data	Data to be encrypted
key	Key for encryption

Returns

Returns decrypted data

See Also

ModelPC::start, ModelPC::inject, ModelPC::unzip

Definition at line 659 of file modelpc.cpp.

Here is the call graph for this function:
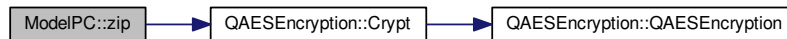
Here is the caller graph for this function:
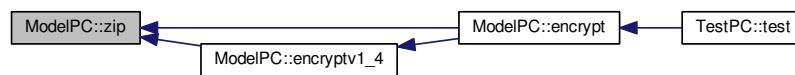
7.4.5 Member Data Documentation

7.4.5.1 QString ModelPC::defaultJPHSDir

defaultJPHSDir Default JPHS directory

Definition at line 91 of file modelpc.h.

7.4.5.2 QString∗ ModelPC::error [protected]

error Current error

Definition at line 104 of file modelpc.h.

7.4.5.3 bool ModelPC::success

success Flag that true by default, but in case of error or cancelling of ProgressDialog it turns to false, which stops execution of ModelPC::circuit

Definition at line 79 of file modelpc.h.

7.4.5.4 long ModelPC::version

version Version of the class

Definition at line 83 of file modelpc.h.

7.4.5.5 QString ModelPC::versionString

versionString Version as string

Definition at line 87 of file modelpc.h.

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

- modelpc.h
- modelpc.cpp

7.5 QAESEncryption Class Reference

The QAESEncryption class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: `https://github.-com/bricke/Qt-AES`.

`#include <qaesencryption.h>`

Inheritance diagram for QAESEncryption:
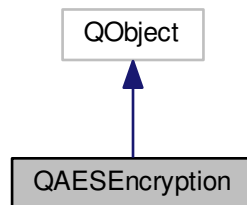
Collaboration diagram for QAESEncryption:
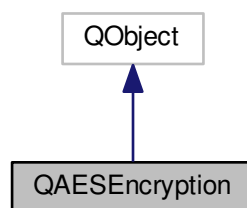
Public Types

- enum Aes { AES_128, AES_192, AES_256 }

    The Aes enum AES Level AES Levels The class supports all AES key lenghts.
- enum Mode { ECB, CBC, CFB, OFB }

    The Mode enum AES Mode The class supports the following operating modes ECB CBC CFB OFB.
- enum Padding { ZERO, PKCS7, ISO }

    The Padding enum Padding By default the padding method is ISO, however, the class supports:

Public Member Functions

- QAESEncryption (QAESEncryption::Aes level, QAESEncryption::Mode mode, QAESEncryption::Padding padding=QAESEncryption::ISO)
- QByteArray encode (const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL)

  encode Encodes data with AES
- QByteArray decode (const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL)

  decode Decodes data with AES
- QByteArray removePadding (const QByteArray &rawText)

  RemovePadding Removes padding.
- QByteArray expandKey (const QByteArray &key)

  ExpandKey Expands the key.

Static Public Member Functions

- static QByteArray Crypt (QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, QAESEncryption::Padding padding=QAESEncryption::ISO)

  Crypt Static encode function.
- static QByteArray Decrypt (QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key, const QByteArray &iv=NULL, QAESEncryption::Padding padding=QAESEncryption::ISO)

  Decrypt Static decode function.
- static QByteArray ExpandKey (QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &key)

  ExpandKey Expands the key.
- static QByteArray RemovePadding (const QByteArray &rawText, QAESEncryption::Padding padding)

  RemovePadding Removes padding.

7.5.1 Detailed Description

The QAESEncryption class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: `https://github.-com/bricke/Qt-AES`.

Author

   Bricke (Matteo B)

Definition at line 14 of file qaesencryption.h.

7.5.2 Member Enumeration Documentation

7.5.2.1 enum **QAESEncryption::Aes**

The Aes enum AES Level AES Levels The class supports all AES key lenghts.

AES_128 AES_192 AES_256

Enumerator

   AES_128

   AES_192

   AES_256

Definition at line 27 of file qaesencryption.h.

7.5.2.2 enum QAESEncryption::Mode

The Mode enum AES Mode The class supports the following operating modes ECB CBC CFB OFB.

Enumerator

>**ECB**
>
>**CBC**
>
>**CFB**
>
>**OFB**

Definition at line 40 of file qaesencryption.h.

7.5.2.3 enum QAESEncryption::Padding

The Padding enum Padding By default the padding method is ISO, however, the class supports:

ZERO PKCS7 ISO

Enumerator

>**ZERO**
>
>**PKCS7**
>
>**ISO**

Definition at line 55 of file qaesencryption.h.

7.5.3 Constructor & Destructor Documentation

7.5.3.1 QAESEncryption::QAESEncryption (QAESEncryption::Aes *level,* QAESEncryption::Mode *mode,* QAESEncryption::Padding *padding =* QAESEncryption::ISO)

Definition at line 67 of file qaesencryption.cpp.

Here is the caller graph for this function:
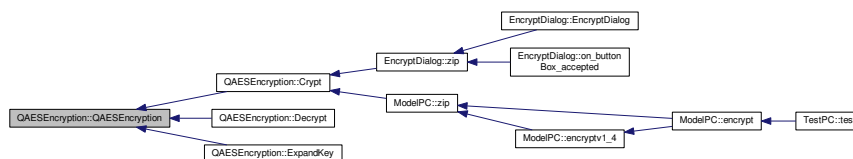
7.5.4 Member Function Documentation

7.5.4.1 QByteArray QAESEncryption::Crypt (QAESEncryption::Aes *level,* QAESEncryption::Mode *mode,* const QByteArray & *rawText,* const QByteArray & *key,* const QByteArray & *iv =* NULL*,* QAESEncryption::Padding *padding =* QAESEncryption::ISO) `[static]`

Crypt Static encode function.

Parameters

level	AES level of encryption
mode	AES mode
rawText	Input data
key	Key for encrytion
iv	IV vector
padding	Padding

Returns

Returns encrypted data

See Also

QAESEncryption::encode, QAESEncryption::Decrypt

Definition at line 6 of file qaesencryption.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:
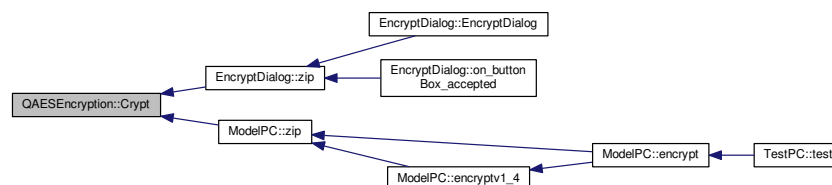
7.5.4.2 QByteArray QAESEncryption::decode (const QByteArray & *rawText,* const QByteArray & *key,* const QByteArray & *iv* = NULL)

decode Decodes data with AES

Note

Basically the non-static method of QAESEncryption::Decrypt

Parameters

rawText	Input data
key	Key
iv	IV vector

Returns

Returns decoded data

See Also

QAESEncryption::Decrypt, QAESEncryption::encode

Definition at line 441 of file qaesencryption.cpp.

Here is the call graph for this function:
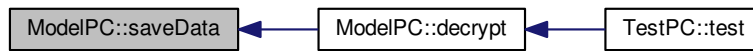

Here is the caller graph for this function:

**7.5.4.3 QByteArray QAESEncryption::Decrypt (QAESEncryption::Aes *level,* QAESEncryption::Mode *mode,* const QByteArray & *rawText,* const QByteArray & *key,* const QByteArray & *iv =* NULL*,* QAESEncryption::Padding *padding =* QAESEncryption::ISO) ** [static]

Decrypt Static decode function.

Parameters

level	AES level of encryption
mode	AES mode
rawText	Encrypted data
key	Key for encrytion
iv	IV vector
padding	Padding

Returns

Returns Decrypted data

See Also

QAESEncryption::decode, QAESEncryption::Crypt

Definition at line 12 of file qaesencryption.cpp.

Here is the call graph for this function:

```
┌──────────────────────────┐      ┌─────────────────────────────────────┐
│ QAESEncryption::Decrypt  │─────▶│ QAESEncryption::QAESEncryption      │
└──────────────────────────┘      └─────────────────────────────────────┘
```

7.5.4.4 QByteArray QAESEncryption::encode (const QByteArray & *rawText,* const QByteArray & *key,* const QByteArray & *iv =* `NULL`)

encode Encodes data with AES

Note

Basically the non-static method of QAESEncryption::Crypt

Parameters

rawText	Input data
key	Key
iv	IV vector

Returns

Returns encoded data

See Also

QAESEncryption::Crypt, QAESEncryption::decode

Definition at line 391 of file qaesencryption.cpp.

Here is the call graph for this function:

```
┌──────────────────────────┐      ┌─────────────────────────────────────┐
│ QAESEncryption::encode   │─────▶│ QAESEncryption::expandKey           │
└──────────────────────────┘      └─────────────────────────────────────┘
```

7.5.4.5 QByteArray QAESEncryption::ExpandKey (QAESEncryption::Aes *level,* QAESEncryption::Mode *mode,* const QByteArray & *key*) `[static]`

ExpandKey Expands the key.

Parameters

level	AES level
mode	AES Mode
key	key

Returns

Returns expanded key (I guess)

See Also

QAESEncryption::expandKey

Definition at line 18 of file qaesencryption.cpp.

Here is the call graph for this function:

7.5.4.6 QByteArray QAESEncryption::expandKey (const QByteArray & key)

ExpandKey Expands the key.

Note

Basically the non-static method of QAESEncryption::ExpandKey

Parameters

key	key

Returns

Returns expanded key (I guess)

See Also

QAESEncryption::ExpandKey

Definition at line 132 of file qaesencryption.cpp.

Here is the caller graph for this function:

7.5.4.7 QByteArray QAESEncryption::RemovePadding (const QByteArray & *rawText,* **QAESEncryption::Padding** *padding* **)** `[static]`

RemovePadding Removes padding.

7.5.4.7 QByteArray QAESEncryption::RemovePadding (const QByteArray & *rawText,* **QAESEncryption::Padding** *padding* **)** `[static]`

Generated on Tue Jan 1 2019 19:38:08 for PictureCrypt by Doxygen

Parameters

rawText	Input data
padding	Padding

Returns

Returns data with removed padding (I guess)

See Also

QAESEncryption::removePadding

Definition at line 23 of file qaesencryption.cpp.

7.5.4.8 QByteArray QAESEncryption::removePadding (const QByteArray & *rawText*)

RemovePadding Removes padding.

Note

Basically the non-static method of QAESEncryption::RemovePadding

Parameters

rawText	Input data

Returns

Returns data with removed padding (I guess)

See Also

QAESEncryption::RemovePadding

Definition at line 490 of file qaesencryption.cpp.

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

- qaesencryption.h
- qaesencryption.cpp

7.6 TestPC Class Reference

The TestPC class AutoTest for ModelPC Currently used in main.cpp.

```
#include <testpc.h>
```

Inheritance diagram for TestPC:
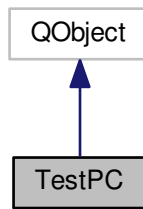
Collaboration diagram for TestPC:
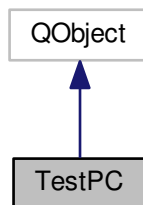
Public Slots

- int startTest ()

    TestPC::startTest Starts the tests running.

Public Member Functions

- TestPC ()

    TestPC::TestPC Constructor.

Static Public Member Functions

- static int Test ()

    TestPC::Test Static function of testing.

Protected Slots

- bool test (QByteArray data, QImage rImage, QString expectedOutput="ok", int mode=0, QString key="", int bitsUsed=8)

    TestPC::test Function calling TestPC::model for tests.

7.6.1 Detailed Description

The TestPC class AutoTest for ModelPC Currently used in main.cpp.

Definition at line 22 of file testpc.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 TestPC::TestPC ( )

TestPC::TestPC Constructor.

Definition at line 5 of file testpc.cpp.

Here is the caller graph for this function:
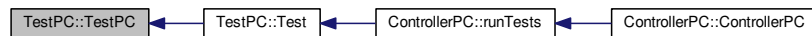
7.6.3 Member Function Documentation

7.6.3.1 int TestPC::startTest ( ) `[slot]`

TestPC::startTest Starts the tests running.

Note

Tests are configured in tests.json

Returns

Returns success of all tests

See Also

TestPC::autoTests

Definition at line 52 of file testpc.cpp.

7.6.3.2 int TestPC::Test ( ) `[static]`

TestPC::Test Static function of testing.

Returns

> Returns result of the testing

Definition at line 13 of file testpc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:
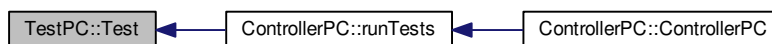
7.6.3.3 **bool TestPC::test (QByteArray** *data,* **QImage** *rImage,* **QString** *expectedOutput =* `"ok"`*,* **int** *mode =* `0`*,* **QString** *key =* `""`*,* **int** *bitsUsed =* `8` **)** `[protected]`,`[slot]`

TestPC::test Function calling TestPC::model for tests.

Parameters

data	Data for test
rImage	Image for test
expectedOutput	Expected output for test ("ok" if everything is well... ok, else errorcode from ErrorsDict.json)
mode	Mode for embedding
key	Key for for test
bitsUsed	Bits Used

Returns

> Returns if test is successful

See Also

TestPC::autoTest, ModelPC::start, ModelPC::decrypt

Definition at line 28 of file testpc.cpp.

Here is the call graph for this function:
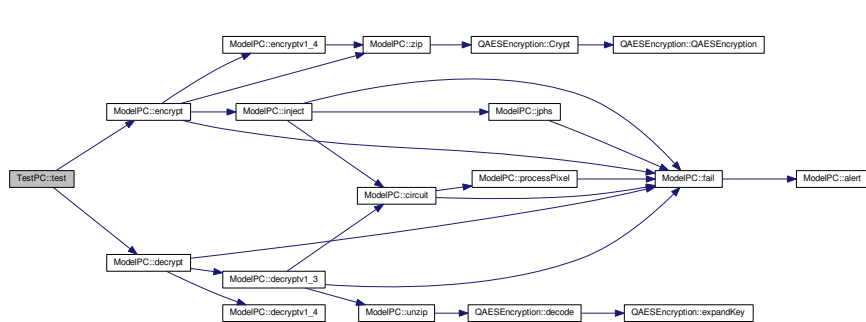
The documentation for this class was generated from the following files:

- testpc.h

- testpc.cpp

7.7 ViewPC Class Reference

The ViewPC class View layer of the app. Controls EncryptDialog and ProgressDialog.

```
#include <viewpc.h>
```

Inheritance diagram for ViewPC:
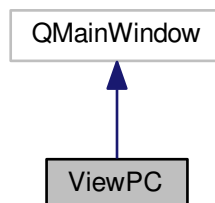
Collaboration diagram for ViewPC:
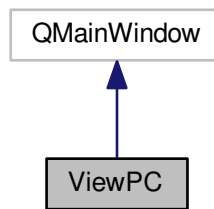
Public Slots

- void alert (QString message, bool isWarning=false)

    ViewPC::alert Slot to create QMessageBox with message.
- void saveData (QByteArray Edata)

    ViewPC::saveData Slot to be called to save data using QFileDialog.
- void saveImage (QImage ∗image)

    ViewPC::saveImage Slot to be called to save image using QFileDialog.
- void setProgress (int val)

    ViewPC::setProgress Slot to set the value of the ProgressDialog (ViewPC::dialog).
- void abortCircuit ()

    ViewPC::abortCircuit Slot to close ProgressDialog (ViewPC::dialog)
- void setEncryptMode (bool encr)

    ViewPC::setEncryptMode Set the encrpt mode (ViewPC::isEncrypt)
- void setVersion (QString version)

    ViewPC::setVersion Set the version of the app from ControllerPC.

Signals

- void encrypt (QByteArray data, QImage ∗image, int mode, int bitsUsed)

    encrypt Signal calling ModelPC::encrypt
- void decrypt (QImage ∗_image, QString key, int mode)

    decrypt Signal calling ModelPC::decrypt
- void abortModel ()

    abortModel Signal calling to stop ModelPC::circuit
- void setJPHSDir (QString dir)

    setJPHSPath Sets the default JPHS directory
- void runTests ()

    runTests Runs tests in ControllerPC via TestPC

Public Member Functions

- ViewPC (QWidget ∗parent=nullptr)
- ∼ViewPC ()

    ViewPC::∼ViewPC Simple destructor for this layer.

Public Attributes

- QProgressDialog ∗ dialog

    dialog ProgressDialog used.
- bool progressDialogClosed

    progressDialogClosed Flag, if dialog is closed.
- QJsonObject errorsDict

Protected Slots

- void on_fileButton_clicked ()

    ViewPC::on_fileButton_clicked Slot to be called, when according button is pressed.
- void on_startButton_clicked ()

    ViewPC::on_startButton_clicked Slot to be called, when Start Button is pressed.
- void on_actionAbout_triggered ()

    ViewPC::on_actionAbout_triggered Opens about page.
- void on_actionHelp_triggered ()

    ViewPC::on_actionHelp_triggered Opens online documentation.

Protected Member Functions

- QString requestKey ()

    ViewPC::requestKey Request keyphrase from user using InputDialog.

7.7.1 Detailed Description

The ViewPC class View layer of the app. Controls EncryptDialog and ProgressDialog.

See Also

ControllerPC, ModelPC, EncryptDialog

Definition at line 35 of file viewpc.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 ViewPC::ViewPC (QWidget ∗ *parent =* `nullptr`) `[explicit]`

Definition at line 4 of file viewpc.cpp.

Here is the call graph for this function:

7.7.2.2 ViewPC::∼ViewPC ( )

ViewPC::∼ViewPC Simple destructor for this layer.

Definition at line 28 of file viewpc.cpp.

7.7.3 Member Function Documentation

7.7.3.1 void ViewPC::abortCircuit ( ) [slot]

ViewPC::abortCircuit Slot to close ProgressDialog (ViewPC::dialog)

Definition at line 222 of file viewpc.cpp.

Here is the caller graph for this function:

7.7.3.2 void ViewPC::abortModel ( ) [signal]

abortModel Signal calling to stop ModelPC::circuit

Here is the caller graph for this function:
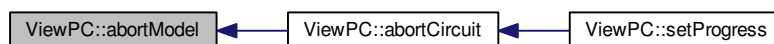
7.7.3.3 void ViewPC::alert (QString *message,* bool *isWarning =* false) [slot]

ViewPC::alert Slot to create QMessageBox with message.

Parameters

message	Message to be shown
isWarning	Flag, if message is critical.

Definition at line 136 of file viewpc.cpp.

Here is the caller graph for this function:
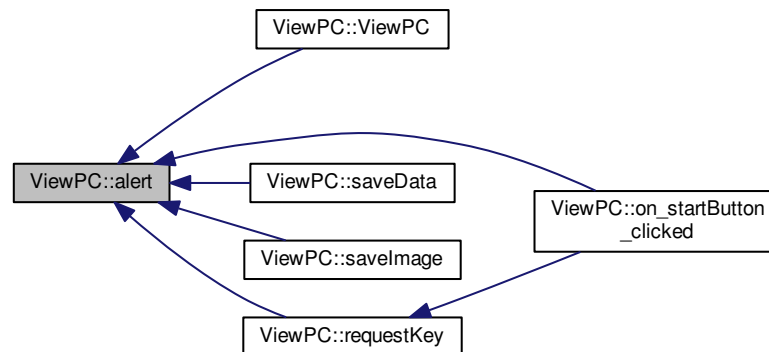
7.7.3.4 void ViewPC::decrypt (QImage ∗ _image, QString key, int mode) `[signal]`

decrypt Signal calling ModelPC::decrypt

Parameters

_image	Image for decryption
key	encryption key // FIXME add param

Here is the caller graph for this function:
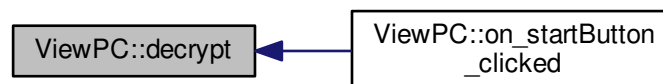
7.7.3.5 void ViewPC::encrypt (QByteArray data, QImage ∗ image, int mode, int bitsUsed) `[signal]`

encrypt Signal calling ModelPC::encrypt

Parameters

data	Data to write
image	Image to be encrypted into.
mode	Mode of encryption
bitsUsed	Bits used per byte

Here is the caller graph for this function:
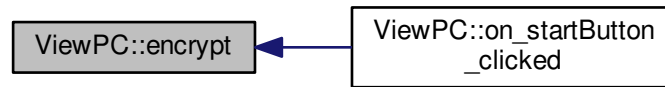
7.7.3.6 void ViewPC::on_actionAbout_triggered ( ) `[protected],[slot]`

ViewPC::on_actionAbout_triggered Opens about page.

Definition at line 277 of file viewpc.cpp.

Here is the call graph for this function:

7.7.3.7 void ViewPC::on_actionHelp_triggered ( ) `[protected],[slot]`

ViewPC::on_actionHelp_triggered Opens online documentation.

Definition at line 287 of file viewpc.cpp.

7.7.3.8 void ViewPC::on_fileButton_clicked ( ) `[protected],[slot]`

ViewPC::on_fileButton_clicked Slot to be called, when according button is pressed.

Definition at line 47 of file viewpc.cpp.

7.7.3.9 void ViewPC::on_startButton_clicked ( ) `[protected],[slot]`

ViewPC::on_startButton_clicked Slot to be called, when Start Button is pressed.

7.7.4 Encrypting

If Encrypting mode is active the data from text browser or from file from file selector will be opened and checked in size.

Note

> File size limit is 16MB

Then the EncryptDialog opens and image and key is selected. Then the ViewPC::encrypt signal is called to start ModelPC::encrypt

7.7.5 Decrypting

Else, the image from file selector is transmitted to ModelPC::decrypt

Definition at line 69 of file viewpc.cpp.

Here is the call graph for this function:

```
┌──────────────────────┐                        ┌──────────────────┐
│ ViewPC::on_startButton│───────────────────────▶│  ViewPC::alert   │
│      _clicked        │      ┌──────────────────▶│                  │
└──────────────────────┘      │ ViewPC::requestKey│└──────────────────┘
                              └──────────────────┘
```
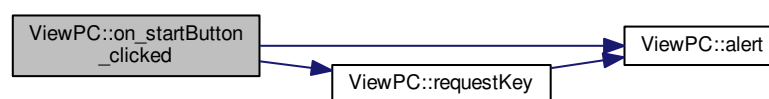
7.7.5.1 QString ViewPC::requestKey ( ) `[protected]`

ViewPC::requestKey Request keyphrase from user using InputDialog.

Returns

> Returns keyphrase

Definition at line 257 of file viewpc.cpp.

Here is the call graph for this function:

```
┌──────────────────────┐          ┌──────────────────┐
│  ViewPC::requestKey  │─────────▶│  ViewPC::alert   │
└──────────────────────┘          └──────────────────┘
```

Here is the caller graph for this function:

```
┌──────────────────────┐          ┌──────────────────────┐
│  ViewPC::requestKey  │◀─────────│ ViewPC::on_startButton│
└──────────────────────┘          │       _clicked       │
                                  └──────────────────────┘
```
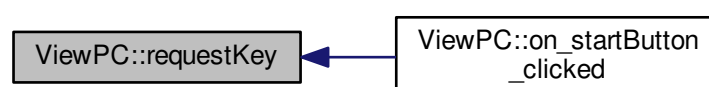
7.7.5.2   void ViewPC::runTests ( ) `[signal]`

runTests Runs tests in ControllerPC via TestPC

7.7.5.3   void ViewPC::saveData (QByteArray *Edata*) `[slot]`

ViewPC::saveData Slot to be called to save data using QFileDialog.

Parameters

Edata	Encrypted data to be saved.

See Also

    ModelPC::encrypt

Definition at line 157 of file viewpc.cpp.

Here is the call graph for this function:

7.7.5.4   void ViewPC::saveImage (QImage ∗ *image*) `[slot]`

ViewPC::saveImage Slot to be called to save image using QFileDialog.

Parameters

image	Image to be saved.

See Also

    ModelPC::decrypt

Definition at line 178 of file viewpc.cpp.

Here is the call graph for this function:
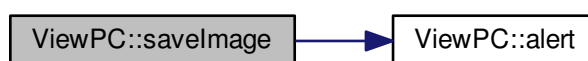
7.7.5.5   void ViewPC::setEncryptMode (bool *encr*)   `[slot]`

ViewPC::setEncryptMode Set the encrpt mode (ViewPC::isEncrypt)

7.7.5.5   void ViewPC::setEncryptMode (bool *encr*)   `[slot]`

Parameters

encr	

Definition at line 235 of file viewpc.cpp.

7.7.5.6 **void ViewPC::setJPHSDir (QString *dir*)** `[signal]`

setJPHSPath Sets the default JPHS directory

Parameters

dir	Directory

7.7.5.7 **void ViewPC::setProgress (int *val*)** `[slot]`

ViewPC::setProgress Slot to set the value of the ProgressDialog (ViewPC::dialog).

Parameters

val	New value of the dialog. If -1, creates ProgressDialog, if 101 closes the dialog.

See Also

> ViewPC::abortCircuit(), ModelPC::setProgress()

Definition at line 196 of file viewpc.cpp.

Here is the call graph for this function:

7.7.5.8 **void ViewPC::setVersion (QString *version*)** `[slot]`

ViewPC::setVersion Set the version of the app from ControllerPC.

Parameters

version	Version as QString

Definition at line 248 of file viewpc.cpp.

Here is the caller graph for this function:

7.7.6 Member Data Documentation

7.7.6.1 QProgressDialog∗ ViewPC::dialog

dialog ProgressDialog used.

See Also

> ViewPC::setProgress, ViewPC::cancel, ModelPC::setProgress

Definition at line 101 of file viewpc.h.

7.7.6.2 QJsonObject ViewPC::errorsDict

Definition at line 107 of file viewpc.h.

7.7.6.3 bool ViewPC::progressDialogClosed

progressDialogClosed Flag, if dialog is closed.

See Also

> ViewPC::abortCircuit, ViewPC::setProgress

Definition at line 106 of file viewpc.h.

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

- viewpc.h
- viewpc.cpp

Chapter 8

File Documentation

8.1   aboutpc.cpp File Reference

```
#include "aboutpc.h"
#include "ui_aboutpc.h"
```
Include dependency graph for aboutpc.cpp:
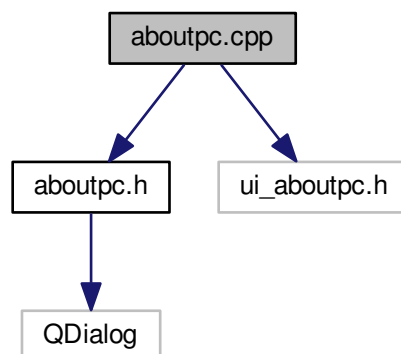
8.2   aboutpc.cpp

```
00001 #include "aboutpc.h"
00002 #include "ui_aboutpc.h"
00003
00004 AboutPC::AboutPC(QWidget *parent) :
00005     QDialog(parent),
00006     ui(new Ui::AboutPC)
00007 {
00008     ui->setupUi(this);
00009 }
00010
00011 AboutPC::~AboutPC()
00012 {
00013     delete ui;
00014 }
00019 void AboutPC::setVersion(QString version)
00020 {
00021     ui->versionLabel->setText("Version " + version);
00022 }
```

8.3 aboutpc.h File Reference

```
#include <QDialog>
```
Include dependency graph for aboutpc.h:
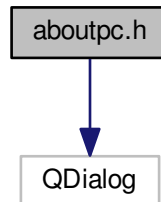
This graph shows which files directly or indirectly include this file:

Classes

- class AboutPC

    The AboutPC class The About Page dialog.

Namespaces

- Ui

8.4 aboutpc.h

```
00001 #ifndef ABOUTPC_H
00002 #define ABOUTPC_H
00003
00004 #include <QDialog>
00005
00006 namespace Ui {
00007 class AboutPC;
00008 }
00012 class AboutPC : public QDialog
00013 {
00014     Q_OBJECT
00015
00016 public:
00017     explicit AboutPC(QWidget *parent = 0);
00018     ~AboutPC();
00019     void setVersion(QString version);
00020
00021 private:
00022     Ui::AboutPC *ui;
00023 };
00024
00025 #endif // ABOUTPC_H
```

8.5 controllerpc.cpp File Reference

```
#include "controllerpc.h"
```
Include dependency graph for controllerpc.cpp:
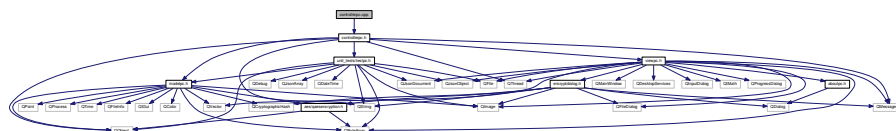
8.6 controllerpc.cpp

```
00001 #include "controllerpc.h"
00002
00009 ControllerPC::ControllerPC()
00010 {
00011     // Layer creation
00012     view = new ViewPC();
00013     model = new ModelPC();
00014     QThread * modelThread = new QThread();
00015     model->moveToThread(modelThread);
00016     modelThread->start();
00017
00018     view->setVersion(model->versionString);
00019     view->show();
00020
00021     // Layers Connection
00022     connect(view, SIGNAL(encrypt(QByteArray,QImage*,int, int)), model, SLOT(inject(QByteArray,QImage*, int,
    int)));
00023     connect(view, SIGNAL(decrypt(QImage*,QString,int)), model, SLOT(decrypt(QImage*, QString, int)));
00024     connect(view, SIGNAL(abortModel()), this, SLOT(abortCircuit()));
00025     connect(view, SIGNAL(setJPHSDir(QString)), this, SLOT(setJPHSDir(QString)));
00026     connect(view, SIGNAL(runTests()), this, SLOT(runTests()));
00027
00028     connect(model, SIGNAL(alertView(QString,bool)), view, SLOT(alert(QString,bool)));
00029     connect(model, SIGNAL(saveData(QByteArray)), view, SLOT(saveData(QByteArray)));
00030     connect(model, SIGNAL(saveImage(QImage*)), view, SLOT(saveImage(QImage*)));
00031     connect(model, SIGNAL(setProgress(int)), view, SLOT(setProgress(int)));
00032 }
00037 void ControllerPC::abortCircuit()
00038 {
00039     model->success = false;
00040 }
00044 void ControllerPC::runTests()
00045 {
00046     // FIXME remove all of this mess and use QtTest
00047     bool res = TestPC::Test();
```

```
00048     QMessageBox o;
00049     o.setText(!res ? "Testing complete! All tests passed." : "Testing failed.");
00050     o.exec();
00051 }
00056 void ControllerPC::setJPHSDir(QString dir)
00057 {
00058     model->defaultJPHSDir = dir;
00059 }
```

8.7 controllerpc.h File Reference

```
#include <QObject>
#include <QString>
#include <QThread>
#include <QMessageBox>
#include <modelpc.h>
#include <viewpc.h>
#include <unit_tests/testpc.h>
```
Include dependency graph for controllerpc.h:
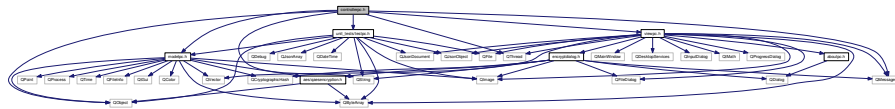
This graph shows which files directly or indirectly include this file:
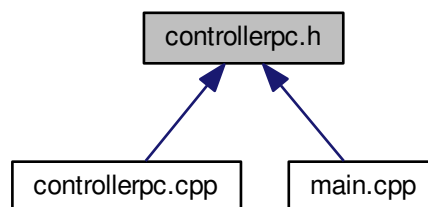
Classes

- class ControllerPC

    The *ControllerPC* class Controller class, which controls View and Model layers.

8.7.1 Detailed Description

Header of ControllerPC class

See Also

    ControllerPC, ModelPC, ViewPC

Definition in file controllerpc.h.

8.8 controllerpc.h

```
00001 #ifndef CONTROLLERPC_H
00002 #define CONTROLLERPC_H
00003
00004 #include <QObject>
00005 #include <QString>
00006 #include <QThread>
00007 #include <QMessageBox>
00008
00009 #include <modelpc.h>
00010 #include <viewpc.h>
00011 #include <unit_tests/testpc.h>
00021 class ControllerPC : public QObject
00022 {
00023     Q_OBJECT
00024 public:
00025     ControllerPC();
00029     long int version;
00033     QString versionString;
00034 public slots:
00035     void abortCircuit();
00036     void runTests();
00037     void setJPHSDir(QString dir);
00038 private:
00039     ViewPC * view;
00040     ModelPC * model;
00041 };
00042
00043 #endif // CONTROLLERPC_H
```

8.9 encryptdialog.cpp File Reference

```
#include "encryptdialog.h"
#include "ui_encryptdialog.h"
```
Include dependency graph for encryptdialog.cpp:
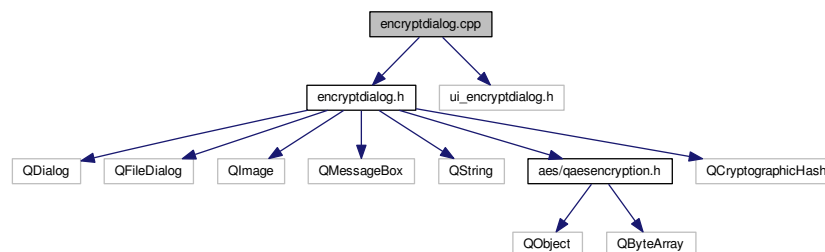
8.10 encryptdialog.cpp

```
00001 #include "encryptdialog.h"
00002 #include "ui_encryptdialog.h"
00009 EncryptDialog::EncryptDialog(QByteArray _data, QWidget *parent) :
00010     QDialog(parent),
00011     ui(new Ui::EncryptDialog)
00012 {
00013     ui->setupUi(this);
00014     data = _data;
00015     success = false;
00016     // UI setup
00017     ui->totalBytes->setText(QString::number(data.size()));
00018     key = "";
00019     compr_data = zip();
00020     long long int compr_data_size = compr_data.size();
00021     ui->zippedBytes->setText(QString::number(compr_data_size));
00022     goodPercentage = false;
00023     bitsUsed = 8;
00024 }
```

```
00025
00026 EncryptDialog::~EncryptDialog()
00027 {
00028     delete ui;
00029 }
00030
00031 void EncryptDialog::alert(QString text)
00032 {
00033     QMessageBox t;
00034     t.setWindowTitle("Message");
00035     t.setIcon(QMessageBox::Warning);
00036     t.setWindowIcon(QIcon(":/mail.png"));
00037     t.setText(text);
00038     t.exec();
00039 }
00046 QByteArray EncryptDialog::zip()
00047 {
00048     // Zip
00049     QByteArray c_data = qCompress(data, 9);
00050     // Encryption
00051     QByteArray hashKey = QCryptographicHash::hash(key.toUtf8(), QCryptographicHash::Sha256);
00052     return QAESEncryption::Crypt(QAESEncryption::AES_256,
    QAESEncryption::ECB, c_data, hashKey);
00053 }
00057 void EncryptDialog::on_fileButton_clicked()
00058 {
00059     // Selet file
00060     inputFileName = QFileDialog::getOpenFileName(this, tr("Open File"), "/", tr("Images (*.png
    *.xpm *.jpg *.jpeg)"));
00061     ui->fileLabel->setText(inputFileName);
00062     // Open image
00063     QImage img(inputFileName);
00064     image = img;
00065     // Get size
00066     size = img.width() * img.height();
00067     // UI setup
00068     long long int compr_data_size = compr_data.size();
00069     ui->zippedBytes->setText(QString::number(compr_data_size));
00070     if(inputFileName.isEmpty()) {
00071         ui->percentage->setText("");
00072         return;
00073     }
00074     double perc = (compr_data_size + 14) * 100 / (size * 3) * bitsUsed / 8;
00075     ui->percentage->setText(QString::number(perc) + "%");
00076     goodPercentage = perc < 70;
00077 }
00082 void EncryptDialog::on_buttonBox_accepted()
00083 {
00084     if(!goodPercentage) {
00085         alert("Your encoding percentage is over 70% which is a bit ambiguous :(");
00086         success = false;
00087         return;
00088     }
00089     // Final zip
00090     key = ui->keyLine->text();
00091     compr_data = zip();
00092     success = true;
00093     close();
00094 }
00098 void EncryptDialog::on_buttonBox_rejected()
00099 {
00100     success = false;
00101     close();
00102 }
00107 void EncryptDialog::on_bitsSlider_valueChanged(int value)
00108 {
00109     bitsUsed = value;
00110     ui->bitsUsedLbl->setText(QString::number(value));
00111     if(ui->percentage->text().isEmpty())
00112         return;
00113     double perc = (compr_data.size() + 14) * 100 / (size * 3) * 8 /
    bitsUsed;
00114     ui->percentage->setText(QString::number(perc) + "%");
00115 }
```

8.11 encryptdialog.h File Reference

```
#include <QDialog>
#include <QFileDialog>
#include <QImage>
#include <QMessageBox>
#include <QString>
#include <aes/qaesencryption.h>
#include <QCryptographicHash>
```
Include dependency graph for encryptdialog.h:
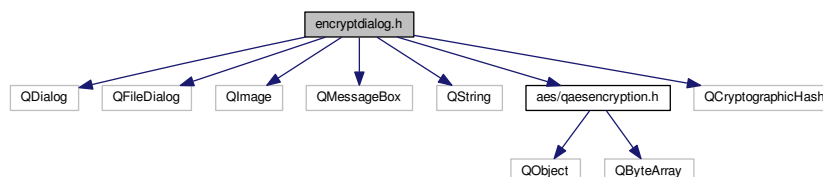
This graph shows which files directly or indirectly include this file:
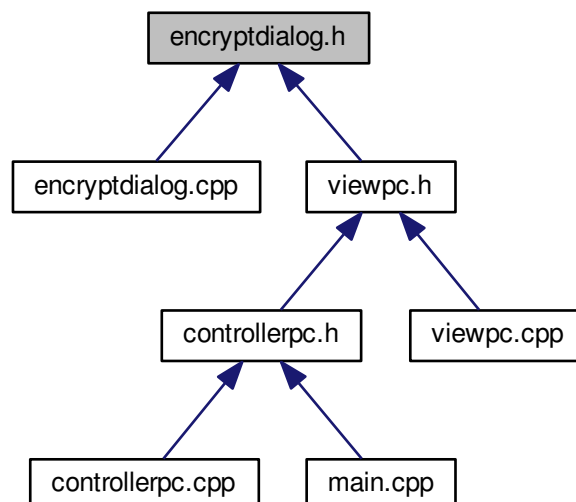
Classes

- class EncryptDialog

  The EncryptDialog class Class to get the image and key to store secret info.
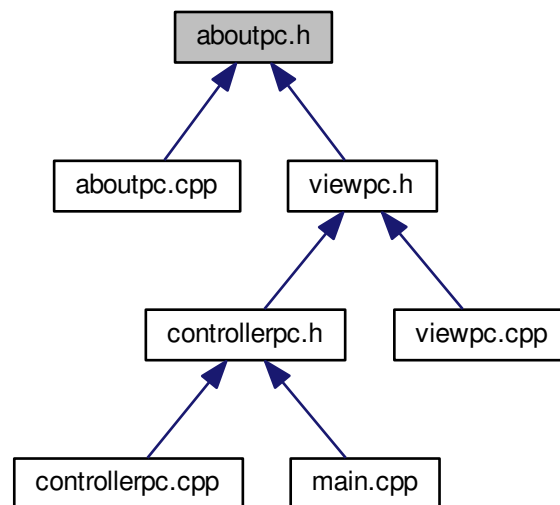
Namespaces

- Ui

8.12 encryptdialog.h

```
00001 #ifndef ENCRYPTDIALOG_H
00002 #define ENCRYPTDIALOG_H
00003
00004 #include <QDialog>
00005 #include <QFileDialog>
00006 #include <QImage>
00007 #include <QMessageBox>
00008 #include <QString>
00009
00010 #include <aes/qaesencryption.h>
00011 #include <QCryptographicHash>
00012
00013 namespace Ui {
00014 class EncryptDialog;
00015 }
00021 class EncryptDialog : public QDialog
00022 {
00023     Q_OBJECT
00024
00025 public:
00026     explicit EncryptDialog(QByteArray _data, QWidget *parent = 0);
00027     ~EncryptDialog();
00028
00029 public slots:
00030     void on_fileButton_clicked();
00031
00032     void on_buttonBox_accepted();
00033
00034     void on_buttonBox_rejected();
00035
00036     void on_bitsSlider_valueChanged(int value);
00037
00038 public:
00042     QByteArray data;
00046     bool success;
00050     QByteArray compr_data;
00054     QString inputFileName;
00058     long long int size;
00062     QString key;
00066     bool goodPercentage;
00070     int val;
00075     int bitsUsed;
00079     QImage image;
00080     QByteArray zip();
00081 private:
00082     Ui::EncryptDialog *ui;
00083     void alert(QString text);
00084 };
00085
00086 #endif // ENCRYPTDIALOG_H
```

8.13 ErrorsDict.json File Reference

8.14 ErrorsDict.json

```
00001 {
00002     "nodata": "No data given!",
00003     "nullimage": "Image not valid!",
00004     "bigkey": "Key is too big, max is 255 bytes!",
00005     "muchdata": "Too much data for this image",
00006     "wrongmode": "Incorrect mode selected",
00007     "wrongimage": "Image wasn't encrypted by this app or is damaged!",
00008     "noreaddata": "Read data is empty!",
00009     "savefilefail": "Cannot save the file!",
00010     "bitsBufferFail": "Something went very wrong! Error code: bitsBuffer",
00011     "nojphs": "JPHS not installed, installation required!\nSee Menu -> Configure -> JPHS directory",
00012     "fail_hash": "Invalid keyphrase"
00013 }
```

8.15 ErrorsDictSetup.py File Reference

Namespaces

- ErrorsDictSetup

Variables

- string ErrorsDictSetup.filename = 'ErrorsDict.json'
- tuple ErrorsDictSetup.raw = open(filename, 'r')
- tuple ErrorsDictSetup.data = json.load(raw)
- tuple ErrorsDictSetup.input_data = input()

8.16 ErrorsDictSetup.py

```
00001 import json
00002 filename = 'ErrorsDict.json'
00003
00004 raw = open(filename, 'r')
00005
00006 data = json.load(raw)
00007 print('Existing data:')
00008 for key, value in data.items():
00009     print(key, value)
00010
00011 print('------------')
00012 print('Type new data')
00013
00014 input_data = input()
00015
00016 while len(input_data):
00017     key, value = map(str, input_data.split('-'))
00018     data[key] = value
00019     input_data = input()
00020
00021 with open(filename, 'w') as f:
00022     json.dump(data, f, indent=4)
```

8.17 main.cpp File Reference

```
#include "controllerpc.h"
#include <QApplication>
```
Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗argv[])

8.17.1 Function Documentation

8.17.1.1 int main (int *argc,* char ∗ *argv[]*)

Definition at line 116 of file main.cpp.

8.18 main.cpp

```
00001 #include "controllerpc.h"
00002 #include <QApplication>
00116 int main(int argc, char *argv[])
00117 {
00118     QApplication a(argc, argv);
00119     ControllerPC w;
00120
00121     return a.exec();
00122 }
```

8.19 modelpc.cpp File Reference

#include "modelpc.h"

#include <QDebug>

#include <QtMath>

Include dependency graph for modelpc.cpp:
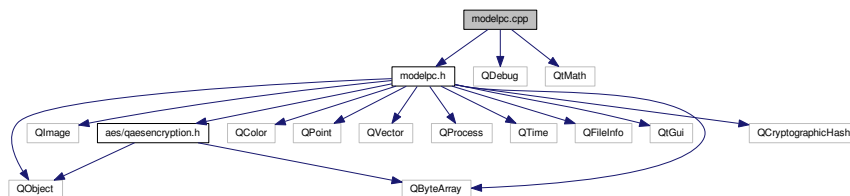
8.20 modelpc.cpp

```
00001 #include "modelpc.h"
00002 #include <QDebug>
00003 #include <QtMath>
00009 ModelPC::ModelPC()
00010 {
00011     // Version control
00012     versionString = "1.3.7";
00013
00014     auto ver = versionString.split(".");
00015     version = ver[0].toInt() * qPow(2, 16) + ver[1].toInt() * qPow(2, 8) + ver[2].toInt();
00016
00017     ver_byte = bytes(ver[0].toInt()) +
00018             bytes(ver[1].toInt()) +
00019             bytes(ver[2].toInt());
00020     // Random seed
00021     qsrand(randSeed());
00022 }
00023
00024 QImage *ModelPC::Encrypt(QByteArray data, QImage *image, int _mode, QString key, int
    _bitsUsed, QString *_error)
00025 {
00026     return ModelPC().encrypt(data, image, _mode, key, _bitsUsed, _error);
00027 }
00028
00029 QImage *ModelPC::Inject(QByteArray encr_data, QImage *image, int _mode, int _bitsUsed,
    QString *_error)
00030 {
00031     return ModelPC().inject(encr_data, image, _mode, _bitsUsed, _error);
00032 }
00033
00034 QByteArray ModelPC::Decrypt(QImage *image, QString key, int _mode, QString *_error)
00035 {
00036     return ModelPC().decrypt(image, key, _mode, _error);
00037 }
00051 QImage * ModelPC::encrypt(QByteArray data, QImage * image, int _mode, QString key, int
    _bitsUsed, QString *_error)
00052 {
00053     // FIXME check for errors
00054     CryptMode mode = CryptMode(_mode);
```

```
00055      // Error management
00056      if(_error == nullptr)
00057          _error = new QString();
00058      *_error = "ok";
00059      error = _error;
00060
00061      if(data.isEmpty()) {
00062          fail("nodata");
00063          return nullptr;
00064      }
00065      if(image == nullptr || image->isNull()) {
00066          fail("nullimage");
00067          return nullptr;
00068      }
00069      if(_bitsUsed < 1 || _bitsUsed > 8) {
00070          fail("bitsWrong");
00071          return nullptr;
00072      }
00073      if(key.isEmpty()) {
00074          fail("no_key");
00075          return nullptr;
00076      }
00077      else if(key.size() > 255) {
00078          fail("bigkey");
00079          return nullptr;
00080      }
00081      if(mode == CryptMode::NotDefined) {
00082          fail("undefined_mode");
00083          return nullptr;
00084      }
00085      long long usedBytes = data.size() + 14 + key.size();
00086      long long size = image->width() * image->height();
00087      if(usedBytes * 100 / (size * 3) * 8 / _bitsUsed > 70) {
00088          fail("muchdata");
00089          return nullptr;
00090      }
00091
00092      switch(mode)
00093      {
00094          case v1_3:
00095          {
00096              QByteArray zipped_data = zip(data, key.toUtf8());
00097              QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
00098              QByteArray encr_data = hash + zipped_data;
00099              if(*error == "ok")
00100                  return inject(encr_data, image, _mode, _bitsUsed, error);
00101              else
00102                  return nullptr;
00103              break;
00104          }
00105          case v1_4:
00106              bitsUsed = _bitsUsed;
00107              encryptv1_4(image, data, key);
00108              return image;
00109          break;
00110          case jphs_mode:
00111              // TODO add jphs
00112              return nullptr;
00113          break;
00114          default:
00115              fail("wrongmode");
00116              return nullptr;
00117      }
00118 }
00119
00130 QImage * ModelPC::inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed,
      QString *_error)
00131 {
00132      CryptMode mode = CryptMode(_mode);
00133      // Error management
00134      if(_error == nullptr)
00135          _error = new QString();
00136      *_error = "ok";
00137      error = _error;
00138
00139      bitsUsed = _bitsUsed;
00140
00141      if(encr_data.isEmpty()) {
00142          fail("nodata");
00143          return nullptr;
00144      }
00145      if(image == nullptr || image->isNull()) {
00146          fail("nullimage");
00147          return nullptr;
00148      }
00149      if(_bitsUsed < 1 || _bitsUsed > 8) {
00150          fail("bitsWrong");
```

```
00151        return nullptr;
00152    }
00153    if(mode == CryptMode::NotDefined) {
00154        fail("undefined_mode");
00155        return nullptr;
00156    }
00157
00158    encr_data = ver_byte + encr_data;
00159    long long int countBytes = encr_data.size();
00160    switch(mode)
00161    {
00162    case v1_3:
00163        circuit(image, &encr_data, countBytes);
00164        break;
00165    case jphs_mode:
00166        jphs(image, &encr_data);
00167        break;
00168    case v1_4:
00169        fail("inject-v1.4");
00170        return nullptr;
00171        break;
00172    default:
00173        fail("wrongmode");
00174        return nullptr;
00175    }
00176
00177
00178    // Saving
00179    if(success) {
00180        emit saveImage(image);
00181        return image;
00182    }
00183    else
00184        return nullptr;
00185 }
00195 QByteArray ModelPC::decrypt(QImage * image, QString key, int _mode, QString *_error)
00196 {
00197    CryptMode mode = CryptMode(_mode);
00198    // Error management
00199    if(_error == nullptr)
00200        _error = new QString();
00201    *_error = "ok";
00202    error = _error;
00203    if(image == nullptr || image->isNull()) {
00204        fail("nullimage");
00205        return nullptr;
00206    }
00207    QByteArray result;
00208
00209    switch (mode) {
00210    case v1_3:
00211        result = decryptv1_3(image, key);
00212    break;
00213    case v1_4:
00214        result = decryptv1_4(image, key);
00215    break;
00216    case jphs_mode:
00217        // TODO add jphs support
00218    break;
00219    case NotDefined:
00220        // TODO check all upper functions
00221    break;
00222    default:
00223        // For invalid modes
00224        fail("wrongmode");
00225        return nullptr;
00226    }
00227    if(*error == "ok")
00228        emit saveData(result);
00229    return result;
00230 }
00235 void ModelPC::fail(QString message)
00236 {
00237    *error = message;
00238    alert(message, true);
00239    success = false;
00240    emit setProgress(101);
00241 }
00247 void ModelPC::jphs(QImage *image, QByteArray *data)
00248 {
00249    // Under Development
00250    return;
00251
00252    // Dead code
00253
00254    success = true;
00255    bool isEncrypt = !data->isEmpty();
```

```
00256       QString targetEXE = defaultJPHSDir + (isEncrypt ? "/jphide.exe" : "/jpseek.exe");
00257       if(!fileExists(targetEXE))
00258       {
00259           fail("nojphs");
00260           return;
00261       }
00262
00263       QString randomFileName = defaultJPHSDir + "/";
00264       qsrand(randSeed());
00265       for(int i = 0; i < 10; i++)
00266           randomFileName.append(97 + qrand() % 25);
00267       image->save(randomFileName + ".jpg");
00268       if(isEncrypt) {
00269           QFile file(randomFileName + ".pc");
00270           if(!file.open(QFile::WriteOnly)) {
00271               fail("savefilefail");
00272               return;
00273           }
00274           file.write(*data);
00275           file.close();
00276
00277           QStringList args;
00278           args << (randomFileName + ".jpg") << (randomFileName + "_out.jpg") << (randomFileName + ".pc");
00279           QProcess prog(this);
00280           prog.start(targetEXE, args);
00281           prog.waitForStarted();
00282           prog.write("test\n");
00283           prog.waitForBytesWritten();
00284           prog.write("test\n");
00285           prog.waitForBytesWritten();
00286           prog.waitForReadyRead();
00287           QByteArray bytes = prog.readAll();
00288           prog.waitForFinished();
00289           //QByteArray readData = prog.readAll();
00290           prog.close();
00291           // Cleaning - Deleting temp files
00292
00293       }
00294       else {
00295
00296       }
00297
00298 }
00299
00308 void ModelPC::circuit(QImage *image, QByteArray *data, long long countBytes)
00309 {
00310       // Some flags and creation of the ProgressDialog
00311       success = true;
00312       emit setProgress(-1);
00313       bool isEncrypt = !data->isEmpty();
00314
00315       // Image setup
00316       int w = image->width();
00317       int h = image->height();
00318
00319       // Visited pixels array
00320       QVector <QPoint> were;
00321       were.push_back(QPoint(0, 0));
00322       were.push_back(QPoint(0, h - 1));
00323       were.push_back(QPoint(w - 1, 0));
00324       were.push_back(QPoint(w - 1, h - 1));
00325
00326       long long int offset = 0;
00327
00328       // Pre-start Cleaning
00329       circuitData = data;
00330       circuitImage = image;
00331       circuitCountBytes = countBytes;
00332       cur = 0;
00333       bitsBuffer.clear();
00334
00335       // Writing Top-Left to Bottom-Left
00336       for(int i = 1; i < h - 1 && mustGoOn(isEncrypt); i++) {
00337           QPoint pos(0, i);
00338           processPixel(pos, &were, isEncrypt);
00339       }
00340       // Writing Bottom-Right to Top-Right
00341       if(mustGoOn(isEncrypt))
00342       {
00343           for(int i = h - 2; i >= 1 && mustGoOn(isEncrypt); i--){
00344               QPoint pos(w - 1, i);
00345               processPixel(pos, &were, isEncrypt);
00346           }
00347       }
00348       // Main cycle
00349       // Strong is considered as actual corner pixel and weak as pixel near it like (1, 0) or (0, 1)
00350       while(mustGoOn(isEncrypt))
```

```
00351    {
00352        // Strong Top-Right to Strong Bottom-Right
00353        for(int i = offset; i < h - offset && mustGoOn(isEncrypt); i++){
00354            QPoint pos(w - offset - 2, i);
00355            processPixel(pos, &were, isEncrypt);
00356        }
00357        // Strong Top-Left to Weak Top-Right
00358        for(int i = offset + 1; i < w - offset - 2 && mustGoOn(isEncrypt); i++){
00359            QPoint pos(i, offset);
00360            processPixel(pos, &were, isEncrypt);
00361        }
00362        // Weak Bottom-Right to Weak Bottom-Left
00363        for(int i = w - 3 - offset; i >= offset + 2 && mustGoOn(isEncrypt); i--){
00364            QPoint pos(i, h - offset - 1);
00365            processPixel(pos, &were, isEncrypt);
00366        }
00367        // Weak Top-Left to Strong Bottom-Left
00368        for(int i = offset + 1; i < h - offset && mustGoOn(isEncrypt); i++){
00369            QPoint pos(offset + 1, i);
00370            processPixel(pos, &were, isEncrypt);
00371        }
00372        offset++;
00373    }
00374    // Extra writing
00375    if(!success)
00376        return;
00377    if(isEncrypt)
00378    {
00379        // Getting past colors
00380        QColor colUL = image->pixelColor(0, 0).toRgb();
00381        QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00382        QColor colDL = image->pixelColor(0, h - 1).toRgb();
00383        QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00384        int red = 0;
00385        int green = 0;
00386        int blue = 0;
00387
00388        // Writing Upper Left
00389        red = (colUL.red() & 224) + (countBytes >> 19);
00390        green = (colUL.green() & 224) + (countBytes >> 14) % 32;
00391        blue = (colUL.blue() & 224) + (countBytes >> 9) % 32;
00392        image->setPixelColor(0, 0, QColor(red, green, blue));
00393
00394        // Writing Upper Right
00395        red = (colUR.red() & 224) + (countBytes >> 4) % 32;
00396        green = (colUR.green() & 224) + ((countBytes % 16) << 1) + 1;
00397        blue = (colUR.blue() & 224) + 9;
00398        image->setPixelColor(w - 1, 0, QColor(red, green, blue));
00399
00400        // Getting extra bytes if left
00401        while(cur < countBytes)
00402            push(mod(circuitData->at(cur++)), 8);
00403        if(bitsBuffer.size() > 20) {
00404            fail("bitsBufferFail");
00405            return;
00406        }
00407        // Getting extra data as long.
00408        long extraData = pop(-2);
00409
00410        // Writing Down Left
00411        red = (colDL.red() & 224) + (extraData >> 15);
00412        green = (colDL.green() & 224) + (extraData >> 10) % 32;
00413        blue = (colDL.blue() & 224) + (extraData >> 5) % 32;
00414        image->setPixelColor(0, h - 1, QColor(red, green, blue));
00415
00416        // Writing Down Right
00417        red = (colDR.red() & 224) + extraData % 32;
00418        green = (colDR.green() & 224);
00419        blue = (colDR.blue() & 224) + ((bitsUsed - 1) << 2) + 2;
00420        image->setPixelColor(w - 1, h - 1, QColor(red, green, blue));
00421    }
00422    else
00423    {
00424        // Read the past pixels
00425        QColor colDL = image->pixelColor(0, h - 1).toRgb();
00426        QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00427
00428        // Read extra data
00429        long extraData = ((colDL.red() % 32) << 15) + ((colDL.green() % 32) << 10);
00430        extraData += ((colDL.blue() % 32) << 5) + colDR.red() % 32;
00431
00432        // Add extra data to the bitsBuffer
00433        push(extraData, (countBytes - cur) * 8 - bitsBuffer.size());
00434
00435        // Move bits from bitsBuffer to the QByteArray
00436        while(!bitsBuffer.isEmpty())
00437            data->append(pop(8));
```

```
00438        }
00439        emit setProgress(101);
00440 }
00441
00449 void ModelPC::processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt)
00450 {
00451        if(!success)
00452             return;
00453        // Check if point was already visited
00454        if(were->contains(pos)){
00455             fail("Point (" + QString::number(pos.x()) + "," + QString::number(pos.y()) + ") was visited
       twice! Error code 2");
00456             return;
00457        }
00458        else
00459             were->push_back(pos);
00460        if(isEncrypt)
00461        {
00462             // Make sure that there are enough bits in bitsBuffer to write
00463             while(bitsBuffer.size() < 3 * bitsUsed)
00464                 push(mod(circuitData->at(cur++)), 8);
00465             // Read past contains
00466             QColor pixelColor = circuitImage->pixelColor(pos);
00467             int red = pixelColor.red();
00468             int green = pixelColor.green();
00469             int blue = pixelColor.blue();
00470
00471             // Write new data in last bitsUsed pixels
00472             red += pop() - red % (int) qPow(2, bitsUsed);
00473             green += pop() - green % (int) qPow(2, bitsUsed);
00474             blue += pop() - blue % (int) qPow(2, bitsUsed);
00475
00476             circuitImage->setPixelColor(pos, QColor(red, green, blue));
00477        }
00478        else
00479        {
00480             QColor read_color = circuitImage->pixelColor(pos).toRgb();
00481             // Reading the pixel
00482             int red = read_color.red();
00483             int green = read_color.green();
00484             int blue = read_color.blue();
00485
00486             // Reading the last bitsUsed pixels
00487             red %= (int) qPow(2, bitsUsed);
00488             green %= (int) qPow(2, bitsUsed);
00489             blue %= (int) qPow(2, bitsUsed);
00490
00491             // Getting the data in the bitsBuffer.
00492             push(red);
00493             push(green);
00494             push(blue);
00495
00496             // Getting data to QByteArray
00497             while(bitsBuffer.size() >= 8) {
00498                 circuitData->append(pop(8));
00499                 cur++;
00500             }
00501        }
00502        emit setProgress(100 * cur / circuitCountBytes);
00503 }
00509 void ModelPC::encryptv1_4(QImage *image, QByteArray data, QString key)
00510 {
00511        QByteArray rand_master = GetRandomBytes(32);
00512        QByteArray pass = QCryptographicHash::hash(key.toUtf8() + rand_master + QString("hi").toUtf8(),
       QCryptographicHash::Sha3_384);
00513        QByteArray noise = GetRandomBytes(data.size() / 10 + 32);
00514        QByteArray bytes_key = GetRandomBytes(32);
00515        QByteArray pass_rand = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_256);
00516        QByteArray zipped = zip(data, pass_rand);
00517 }
00524 QByteArray ModelPC::decryptv1_4(QImage *image, QString key)
00525 {
00526
00527 }
00528
00535 QByteArray ModelPC::decryptv1_3(QImage *image, QString key)
00536 {
00537        // Image opening
00538        int w = image->width();
00539        int h = image->height();
00540
00541        // Getting corner pixels
00542        QColor colUL = image->pixelColor(0, 0).toRgb();
00543        QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00544        QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00545
00546
```

```
00547      // Getting verification code
00548      int verifCode = (((colUR.green() % 2) << 5) + colUR.blue() % 32) << 2;
00549      verifCode += colDR.blue() % 4;
00550      if(verifCode != 166){
00551          fail("veriffail");
00552          return nullptr;
00553      }
00554      // Getting number of bytes
00555      long long int countBytes = (colUL.blue() % 32 + ((colUL.green() % 32) << 5) + ((colUL.red() % 32) << 10
      )) << 9;
00556      countBytes += ((colUR.red() % 32) << 4) + (colUR.green() >> 1) % 16;
00557
00558      bitsUsed = (colDR.blue() >> 2) % 8 + 1;
00559      // FIXME check if works
00560      // curMode = colDR.green() % 32;
00561
00562      // Start of the circuit
00563      QByteArray data;
00564      circuit(image, &data, countBytes);
00565
00566      // Check if circuit was successful
00567      if(!success)
00568          return nullptr;
00569      if(data.isEmpty())
00570      {
00571          fail("noreaddata");
00572          return nullptr;
00573
00574      }
00575      // Version check
00576      long long int _ver = mod(data.at(0)) * qPow(2, 16);
00577      _ver += mod(data.at(1)) * qPow(2, 8);
00578      _ver += mod(data.at(2));
00579      data.remove(0, 3);
00580      if(_ver > version) {
00581          fail("Picture's app version is newer than yours. Image version is "
00582                  + generateVersionString(_ver) + ", yours is "
00583                  + generateVersionString(version) + ".");
00584          return nullptr;
00585      }
00586      else if(_ver < version) {
00587          fail("Picture's app version is older than yours. Image version is "
00588                  + generateVersionString(_ver) + ", yours is "
00589                  + generateVersionString(version) + ".");
00590          return nullptr;
00591      }
00592      // Get the hash
00593      QByteArray hash = data.left(32);
00594      data.remove(0, 32);
00595
00596      // Unzip
00597      QByteArray unzipped_data = unzip(data, key.toUtf8());
00598      QByteArray our_hash = QCryptographicHash::hash(unzipped_data, QCryptographicHash::Sha256);
00599      if(our_hash != hash) {
00600          fail("fail_hash");
00601          return QByteArray("");
00602      }
00603      return unzipped_data;
00604 }
00605 long ModelPC::pop(int bits)
00606 {
00607      // Hard to say
00608      long res = 0;
00609      int poppedBits = bits == -1 ? bitsUsed : bits;
00610      if(bits == -2)
00611          poppedBits = bitsBuffer.size();
00612      for(int i = 0; i < poppedBits; i++)
00613          res += bitsBuffer[i] * qPow(2, poppedBits - i - 1);
00614      bitsBuffer.remove(0, poppedBits);
00615      return res;
00616 }
00617
00618 void ModelPC::push(int data, int bits)
00619 {
00620      // That's easier, but also hard
00621      int buf_size = bitsBuffer.size();
00622      int extraSize = bits == -1 ? bitsUsed : bits;
00623      bitsBuffer.resize(buf_size + extraSize);
00624      for(int i = bitsBuffer.size() - 1; i >= buf_size; i--, data >>= 1)
00625          bitsBuffer[i] = data % 2;
00626 }
00627
00628 bool ModelPC::mustGoOn(bool isEncrypt)
00629 {
00630      return success && (isEncrypt ? (circuitCountBytes - cur) * 8 + bitsBuffer.size() >= bitsUsed * 3
      :
00631                                      circuitData->size() * 8 + bitsBuffer.size() <
```

```
00632                                       circuitCountBytes * 8 - (circuitCountBytes * 8)% (bitsUsed * 3));
00633 }
00642 QByteArray ModelPC::unzip(QByteArray data, QByteArray key)
00643 {
00644     // Decryption
00645     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00646     QAESEncryption encryption(QAESEncryption::AES_256,
      QAESEncryption::ECB);
00647     QByteArray new_data = encryption.decode(data, hashKey);
00648     // Decompressing
00649     return qUncompress(new_data);
00650 }
00659 QByteArray ModelPC::zip(QByteArray data, QByteArray key)
00660 {
00661     // Zip
00662     QByteArray c_data = qCompress(data, 9);
00663     // Encryption
00664     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00665     return QAESEncryption::Crypt(QAESEncryption::AES_256,
      QAESEncryption::ECB, c_data, hashKey);
00666 }
00667
00668 bool ModelPC::fileExists(QString path)
00669 {
00670     QFileInfo check_file(path);
00671     return check_file.exists() && check_file.isFile();
00672 }
00673
00680 QByteArray ModelPC::bytes(long long n)
00681 {
00682     return QByteArray::fromHex(QByteArray::number(n, 16));
00683 }
00690 unsigned int ModelPC::mod(int input)
00691 {
00692     if(input < 0)
00693         return (unsigned int) (256 + input);
00694     else
00695         return (unsigned int) input;
00696 }
00703 void ModelPC::alert(QString message, bool isWarning)
00704 {
00705     emit alertView(message, isWarning);
00706 }
00712 QColor ModelPC::RGBbytes(long long byte)
00713 {
00714     int blue = byte % 256;
00715     int green = (byte / 256) % 256;
00716     int red = byte / qPow(2, 16);
00717     return QColor(red, green, blue);
00718 }
00719
00720 QString ModelPC::generateVersionString(long ver)
00721 {
00722     return QString::number((int)( ver / qPow(2, 16))) + "." + QString::number(((int) (ver / 256)) % 256) +
      "." + QString::number(ver % 256);
00723 }
00724
00725 uint ModelPC::randSeed()
00726 {
00727     QTime time = QTime::currentTime();
00728     uint randSeed = time.msecsSinceStartOfDay() % 65536 + time.minute() * 21 + time.second() * 2;
00729     return randSeed;
00730 }
00731 QByteArray ModelPC::GetRandomBytes(long long count)
00732 {
00733     QByteArray res;
00734     for(int i = 0; i < count; i++)
00735         res += qrand() % 256;
00736     return res;
00737 }
```

8.21 modelpc.h File Reference

```
#include <QObject>
#include <QImage>
#include <QByteArray>
#include <QColor>
#include <QPoint>
#include <QVector>
#include <QProcess>
#include <QTime>
#include <QFileInfo>
#include <QtGui>
#include <aes/qaesencryption.h>
#include <QCryptographicHash>
```

Include dependency graph for modelpc.h:
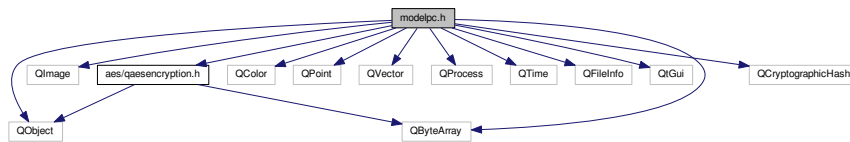
This graph shows which files directly or indirectly include this file:
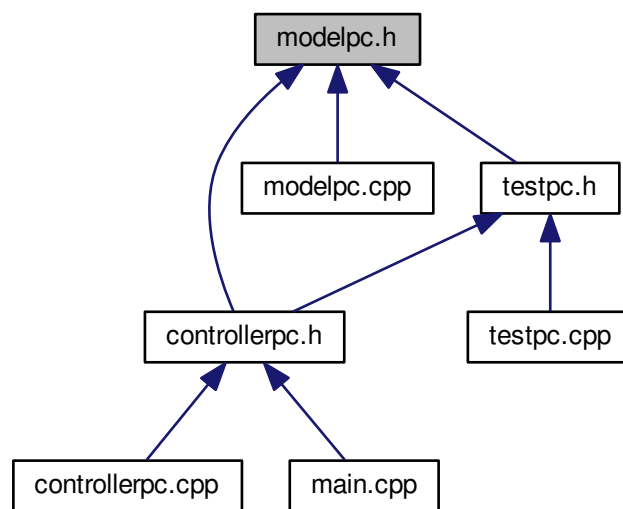
Classes

- class ModelPC

  The ModelPC class Model Layer of the app. Main class that does the work of PictureCrypt logic Controled by ControllerPC.

8.21.1 Detailed Description

Header of ModelPC class

See Also

> ControllerPC, ModelPC, ViewPC

Definition in file modelpc.h.

8.22 modelpc.h

```
00001 #ifndef MODELPC_H
00002 #define MODELPC_H
00003
00004 #include <QObject>
00005 #include <QImage>
00006 #include <QByteArray>
00007 #include <QColor>
00008 #include <QPoint>
00009 #include <QVector>
00010 #include <QProcess>
00011 #include <QTime>
00012 #include <QFileInfo>
00013 #include <QtGui>
00014
00015 #include <aes/qaesencryption.h>
00016 #include <QCryptographicHash>
00017
00030 class ModelPC : public QObject
00031 {
00032     Q_OBJECT
00033 public:
00034     ModelPC();
00035     enum CryptMode {NotDefined, v1_3, v1_4, jphs_mode};
00036     static QImage *Encrypt(QByteArray data, QImage *image, int _mode, QString key = "", int
    _bitsUsed = 8, QString *_error = nullptr);
00037     static QImage *Inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString
    *_error = nullptr);
00038     static QByteArray Decrypt(QImage * image, QString key, int _mode = 0, QString *_error = nullptr)
    ;
00039
00040 signals:
00047     void alertView(QString messageCode, bool isWarning);
00052     void saveData(QByteArray data);
00057     void saveImage(QImage *image);
00063     void setProgress(int val);
00064
00065 public slots:
00066     QImage *encrypt(QByteArray data, QImage *image, int _mode, QString key = "", int _bitsUsed =
    8, QString *_error = nullptr);
00067     QImage *inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString *
    _error = nullptr);
00068     QByteArray decrypt(QImage * image, QString key, int _mode = 0, QString *_error = nullptr);
00069     void fail(QString message);
00070     void alert(QString message, bool isWarning = false);
00071
00072 public:
00073     QByteArray unzip(QByteArray data, QByteArray key);
00074
00079     bool success;
00083     long version;
00087     QString versionString;
00091     QString defaultJPHSDir;
00092 protected:
00093     void circuit(QImage * image, QByteArray * data, long long int countBytes);
00094     void jphs(QImage * image, QByteArray * data);
00095     void processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt);
00096     void encryptv1_4(QImage *image, QByteArray data, QString key);
00097     QByteArray decryptv1_3(QImage * image, QString key);
00098     QByteArray decryptv1_4(QImage * image, QString key);
00099     QByteArray zip(QByteArray data, QByteArray key);
00100
00104     QString * error;
00105 private:
00106     int bitsUsed;
00107     bool fileExists(QString path);
00108     QByteArray bytes(long long n);
00109     unsigned int mod(int input);
00110     QByteArray ver_byte;
```

```
00111    QColor RGBbytes(long long byte);
00112    QString generateVersionString(long ver);
00113    uint randSeed();
00114
00115    QByteArray * circuitData;
00116    QImage * circuitImage;
00117    long long circuitCountBytes;
00118    long cur;
00119    bool mustGoOn(bool isEncrypt);
00120
00121    QVector <bool> bitsBuffer;
00122    long pop(int bits = -1);
00123    void push(int data, int bits = -1);
00124
00125    void setError(QString word);
00126    QByteArray GetRandomBytes(long long count = 32);
00127 };
00128
00129 #endif // MODELPC_H
```

8.23 qaesencryption.cpp File Reference

```
#include "qaesencryption.h"
```
Include dependency graph for qaesencryption.cpp:
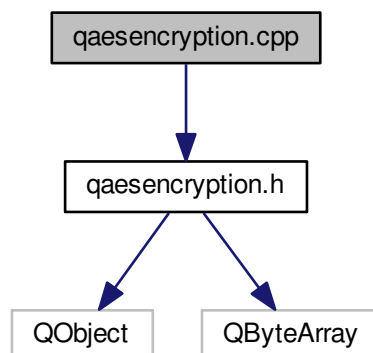
Functions

- quint8 xTime (quint8 x)

- quint8 multiply (quint8 x, quint8 y)

8.23.1 Function Documentation

8.23.1.1 quint8 multiply (quint8 *x,* quint8 *y*) `[inline]`

Definition at line 57 of file qaesencryption.cpp.

Here is the call graph for this function:

8.23.1.2   quint8 xTime (quint8 *x*)   `[inline]`

Definition at line 53 of file qaesencryption.cpp.

Here is the caller graph for this function:

8.24   qaesencryption.cpp

```
00001 #include "qaesencryption.h"
00002
00003 /*
00004  * Static Functions
00005  * */
00006 QByteArray QAESEncryption::Crypt(QAESEncryption::Aes level,
       QAESEncryption::Mode mode, const QByteArray &rawText,
00007                                 const QByteArray &key, const QByteArray &iv,
       QAESEncryption::Padding padding)
00008 {
00009     return QAESEncryption(level, mode, padding).encode(rawText, key, iv);
00010 }
00011
00012 QByteArray QAESEncryption::Decrypt(QAESEncryption::Aes level,
       QAESEncryption::Mode mode, const QByteArray &rawText,
00013                                   const QByteArray &key, const QByteArray &iv,
       QAESEncryption::Padding padding)
00014 {
00015     return QAESEncryption(level, mode, padding).decode(rawText, key, iv);
00016 }
00017
00018 QByteArray QAESEncryption::ExpandKey(
       QAESEncryption::Aes level, QAESEncryption::Mode mode, const
       QByteArray &key)
00019 {
00020     return QAESEncryption(level, mode).expandKey(key);
00021 }
00022
00023 QByteArray QAESEncryption::RemovePadding(const QByteArray &rawText,
       QAESEncryption::Padding padding)
00024 {
00025     QByteArray ret(rawText);
00026     switch (padding)
00027     {
00028     case Padding::ZERO:
00029         //Works only if the last byte of the decoded array is not zero
00030         while (ret.at(ret.length()-1) == 0x00)
```

```
00031                ret.remove(ret.length()-1, 1);
00032            break;
00033        case Padding::PKCS7:
00034            ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00035            break;
00036        case Padding::ISO:
00037            ret.truncate(ret.lastIndexOf(0x80));
00038            break;
00039        default:
00040            //do nothing
00041            break;
00042        }
00043        return ret;
00044 }
00045 /*
00046  * End Static function declarations
00047  * */
00048
00049 /*
00050  * Inline Functions
00051  * */
00052
00053 inline quint8 xTime(quint8 x){
00054    return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
00055 }
00056
00057 inline quint8 multiply(quint8 x, quint8 y){
00058    return (((y & 1) * x) ^ ((y>>1 & 1) * xTime(x)) ^ ((y>>2 & 1) * xTime(
      xTime(x))) ^ ((y>>3 & 1)
00059                * xTime(xTime(xTime(x)))) ^ ((y>>4 & 1) * xTime(
      xTime(xTime(xTime(x)))))));
00060 }
00061
00062 /*
00063  * End Inline functions
00064  * */
00065
00066
00067 QAESEncryption::QAESEncryption(Aes level, Mode mode,
00068                                Padding padding)
00069     : m_nb(4), m_blocklen(16), m_level(level), m_mode(mode), m_padding(padding)
00070 {
00071     m_state = NULL;
00072
00073     switch (level)
00074     {
00075     case AES_128: {
00076         AES128 aes;
00077         m_nk = aes.nk;
00078         m_keyLen = aes.keylen;
00079         m_nr = aes.nr;
00080         m_expandedKey = aes.expandedKey;
00081         }
00082         break;
00083     case AES_192: {
00084         AES192 aes;
00085         m_nk = aes.nk;
00086         m_keyLen = aes.keylen;
00087         m_nr = aes.nr;
00088         m_expandedKey = aes.expandedKey;
00089         }
00090         break;
00091     case AES_256: {
00092         AES256 aes;
00093         m_nk = aes.nk;
00094         m_keyLen = aes.keylen;
00095         m_nr = aes.nr;
00096         m_expandedKey = aes.expandedKey;
00097         }
00098         break;
00099     default: {
00100         AES128 aes;
00101         m_nk = aes.nk;
00102         m_keyLen = aes.keylen;
00103         m_nr = aes.nr;
00104         m_expandedKey = aes.expandedKey;
00105         }
00106         break;
00107     }
00108
00109 }
00110 QByteArray QAESEncryption::getPadding(int currSize, int alignment)
00111 {
00112     int size = (alignment - currSize % alignment) % alignment;
00113     if (size == 0) return QByteArray();
00114     switch(m_padding)
00115     {
```

```
00116      case Padding::ZERO:
00117          return QByteArray(size, 0x00);
00118          break;
00119      case Padding::PKCS7:
00120          return QByteArray(size,size);
00121          break;
00122      case Padding::ISO:
00123          return QByteArray (size-1, 0x00).prepend(0x80);
00124          break;
00125      default:
00126          return QByteArray(size, 0x00);
00127          break;
00128      }
00129      return QByteArray(size, 0x00);
00130 }
00131
00132 QByteArray QAESEncryption::expandKey(const QByteArray &key)
00133 {
00134    int i, k;
00135    quint8 tempa[4]; // Used for the column/row operations
00136    QByteArray roundKey(key);
00137
00138    // The first round key is the key itself.
00139    // ...
00140
00141    // All other round keys are found from the previous round keys.
00142    //i == Nk
00143    for(i = m_nk; i < m_nb * (m_nr + 1); i++)
00144    {
00145      tempa[0] = (quint8) roundKey.at((i-1) * 4 + 0);
00146      tempa[1] = (quint8) roundKey.at((i-1) * 4 + 1);
00147      tempa[2] = (quint8) roundKey.at((i-1) * 4 + 2);
00148      tempa[3] = (quint8) roundKey.at((i-1) * 4 + 3);
00149
00150      if (i % m_nk == 0)
00151      {
00152          // This function shifts the 4 bytes in a word to the left once.
00153          // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]
00154
00155          // Function RotWord()
00156          k = tempa[0];
00157          tempa[0] = tempa[1];
00158          tempa[1] = tempa[2];
00159          tempa[2] = tempa[3];
00160          tempa[3] = k;
00161
00162          // Function Subword()
00163          tempa[0] = getSBoxValue(tempa[0]);
00164          tempa[1] = getSBoxValue(tempa[1]);
00165          tempa[2] = getSBoxValue(tempa[2]);
00166          tempa[3] = getSBoxValue(tempa[3]);
00167
00168          tempa[0] =  tempa[0] ^ Rcon[i/m_nk];
00169      }
00170      if (m_level == AES_256 && i % m_nk == 4)
00171      {
00172          // Function Subword()
00173          tempa[0] = getSBoxValue(tempa[0]);
00174          tempa[1] = getSBoxValue(tempa[1]);
00175          tempa[2] = getSBoxValue(tempa[2]);
00176          tempa[3] = getSBoxValue(tempa[3]);
00177      }
00178      roundKey.insert(i * 4 + 0, (quint8) roundKey.at((i - m_nk) * 4 + 0) ^ tempa[0]);
00179      roundKey.insert(i * 4 + 1, (quint8) roundKey.at((i - m_nk) * 4 + 1) ^ tempa[1]);
00180      roundKey.insert(i * 4 + 2, (quint8) roundKey.at((i - m_nk) * 4 + 2) ^ tempa[2]);
00181      roundKey.insert(i * 4 + 3, (quint8) roundKey.at((i - m_nk) * 4 + 3) ^ tempa[3]);
00182    }
00183    return roundKey;
00184 }
00185
00186 // This function adds the round key to state.
00187 // The round key is added to the state by an XOR function.
00188 void QAESEncryption::addRoundKey(const quint8 round, const QByteArray expKey)
00189 {
00190    QByteArray::iterator it = m_state->begin();
00191    for(int i=0; i < 16; ++i)
00192        it[i] = (quint8) it[i] ^ (quint8) expKey.at(round * m_nb * 4 + (i/4) * m_nb + (i%4));
00193 }
00194
00195 // The SubBytes Function Substitutes the values in the
00196 // state matrix with values in an S-box.
00197 void QAESEncryption::subBytes()
00198 {
00199    QByteArray::iterator it = m_state->begin();
00200    for(int i = 0; i < 16; i++)
00201      it[i] = getSBoxValue((quint8) it[i]);
00202 }
```

```
00203
00204 // The ShiftRows() function shifts the rows in the state to the left.
00205 // Each row is shifted with different offset.
00206 // Offset = Row number. So the first row is not shifted.
00207 void QAESEncryption::shiftRows()
00208 {
00209     QByteArray::iterator it = m_state->begin();
00210     quint8 temp;
00211     //Keep in mind that QByteArray is column-driven!!
00212
00213     //Shift 1 to left
00214     temp  = (quint8)it[1];
00215     it[1]  = (quint8)it[5];
00216     it[5]  = (quint8)it[9];
00217     it[9]  = (quint8)it[13];
00218     it[13] = (quint8)temp;
00219
00220     //Shift 2 to left
00221     temp  = (quint8)it[2];
00222     it[2]  = (quint8)it[10];
00223     it[10] = (quint8)temp;
00224     temp  = (quint8)it[6];
00225     it[6]  = (quint8)it[14];
00226     it[14] = (quint8)temp;
00227
00228     //Shift 3 to left
00229     temp  = (quint8)it[3];
00230     it[3]  = (quint8)it[15];
00231     it[15] = (quint8)it[11];
00232     it[11] = (quint8)it[7];
00233     it[7]  = (quint8)temp;
00234 }
00235
00236 // MixColumns function mixes the columns of the state matrix
00237 //optimized!!
00238 void QAESEncryption::mixColumns()
00239 {
00240   QByteArray::iterator it = m_state->begin();
00241   quint8 tmp, tm, t;
00242
00243   for(int i = 0; i < 16; i += 4){
00244     t       = (quint8)it[i];
00245     tmp     =  (quint8)it[i] ^ (quint8)it[i+1] ^ (quint8)it[i+2] ^ (quint8)it[i+3] ;
00246
00247     tm       = xTime( (quint8)it[i] ^ (quint8)it[i+1] );
00248     it[i]    = (quint8)it[i] ^ (quint8)tm ^ (quint8)tmp;
00249
00250     tm       = xTime( (quint8)it[i+1] ^ (quint8)it[i+2]);
00251     it[i+1] = (quint8)it[i+1] ^ (quint8)tm ^ (quint8)tmp;
00252
00253     tm       = xTime( (quint8)it[i+2] ^ (quint8)it[i+3]);
00254     it[i+2] =(quint8)it[i+2] ^ (quint8)tm ^ (quint8)tmp;
00255
00256     tm       = xTime((quint8)it[i+3] ^ (quint8)t);
00257     it[i+3] =(quint8)it[i+3] ^ (quint8)tm ^ (quint8)tmp;
00258   }
00259 }
00260
00261 // MixColumns function mixes the columns of the state matrix.
00262 // The method used to multiply may be difficult to understand for the inexperienced.
00263 // Please use the references to gain more information.
00264 void QAESEncryption::invMixColumns()
00265 {
00266   QByteArray::iterator it = m_state->begin();
00267   quint8 a,b,c,d;
00268   for(int i = 0; i < 16; i+=4){
00269     a = (quint8) it[i];
00270     b = (quint8) it[i+1];
00271     c = (quint8) it[i+2];
00272     d = (quint8) it[i+3];
00273
00274     it[i]   = (quint8) (multiply(a, 0x0e) ^ multiply(b, 0x0b) ^
      multiply(c, 0x0d) ^ multiply(d, 0x09));
00275     it[i+1] = (quint8) (multiply(a, 0x09) ^ multiply(b, 0x0e) ^
      multiply(c, 0x0b) ^ multiply(d, 0x0d));
00276     it[i+2] = (quint8) (multiply(a, 0x0d) ^ multiply(b, 0x09) ^
      multiply(c, 0x0e) ^ multiply(d, 0x0b));
00277     it[i+3] = (quint8) (multiply(a, 0x0b) ^ multiply(b, 0x0d) ^
      multiply(c, 0x09) ^ multiply(d, 0x0e));
00278   }
00279 }
00280
00281 // The SubBytes Function Substitutes the values in the
00282 // state matrix with values in an S-box.
00283 void QAESEncryption::invSubBytes()
00284 {
00285     QByteArray::iterator it = m_state->begin();
```

```
00286     for(int i = 0; i < 16; ++i)
00287         it[i] = getSBoxInvert((quint8) it[i]);
00288 }
00289
00290 void QAESEncryption::invShiftRows()
00291 {
00292     QByteArray::iterator it = m_state->begin();
00293     uint8_t temp;
00294
00295     //Keep in mind that QByteArray is column-driven!!
00296
00297     //Shift 1 to right
00298     temp  = (quint8)it[13];
00299     it[13] = (quint8)it[9];
00300     it[9]  = (quint8)it[5];
00301     it[5]  = (quint8)it[1];
00302     it[1]  = (quint8)temp;
00303
00304     //Shift 2
00305     temp  = (quint8)it[10];
00306     it[10] = (quint8)it[2];
00307     it[2]  = (quint8)temp;
00308     temp  = (quint8)it[14];
00309     it[14] = (quint8)it[6];
00310     it[6]  = (quint8)temp;
00311
00312     //Shift 3
00313     temp  = (quint8)it[15];
00314     it[15] = (quint8)it[3];
00315     it[3]  = (quint8)it[7];
00316     it[7]  = (quint8)it[11];
00317     it[11] = (quint8)temp;
00318 }
00319
00320 QByteArray QAESEncryption::byteXor(const QByteArray &a, const QByteArray &b)
00321 {
00322   QByteArray::const_iterator it_a = a.begin();
00323   QByteArray::const_iterator it_b = b.begin();
00324   QByteArray ret;
00325
00326   //for(int i = 0; i < m_blocklen; i++)
00327   for(int i = 0; i < std::min(a.size(), b.size()); i++)
00328       ret.insert(i,it_a[i] ^ it_b[i]);
00329
00330   return ret;
00331 }
00332
00333 // Cipher is the main function that encrypts the PlainText.
00334 QByteArray QAESEncryption::cipher(const QByteArray &expKey, const QByteArray &in)
00335 {
00336
00337   //m_state is the input buffer...
00338   QByteArray output(in);
00339   m_state = &output;
00340
00341   // Add the First round key to the state before starting the rounds.
00342   addRoundKey(0, expKey);
00343
00344   // There will be Nr rounds.
00345   // The first Nr-1 rounds are identical.
00346   // These Nr-1 rounds are executed in the loop below.
00347   for(quint8 round = 1; round < m_nr; ++round){
00348     subBytes();
00349     shiftRows();
00350     mixColumns();
00351     addRoundKey(round, expKey);
00352   }
00353
00354   // The last round is given below.
00355   // The MixColumns function is not here in the last round.
00356   subBytes();
00357   shiftRows();
00358   addRoundKey(m_nr, expKey);
00359
00360   return output;
00361 }
00362
00363 QByteArray QAESEncryption::invCipher(const QByteArray &expKey, const QByteArray &in)
00364 {
00365     //m_state is the input buffer.... handle it!
00366     QByteArray output(in);
00367     m_state = &output;
00368
00369     // Add the First round key to the state before starting the rounds.
00370     addRoundKey(m_nr, expKey);
00371
00372     // There will be Nr rounds.
```

```
00373     // The first Nr-1 rounds are identical.
00374     // These Nr-1 rounds are executed in the loop below.
00375     for(quint8 round=m_nr-1; round>0 ; round--){
00376         invShiftRows();
00377         invSubBytes();
00378         addRoundKey(round, expKey);
00379         invMixColumns();
00380     }
00381
00382     // The last round is given below.
00383     // The MixColumns function is not here in the last round.
00384     invShiftRows();
00385     invSubBytes();
00386     addRoundKey(0, expKey);
00387
00388     return output;
00389 }
00390
00391 QByteArray QAESEncryption::encode(const QByteArray &rawText, const QByteArray &key,
      const QByteArray &iv)
00392 {
00393     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00394         return QByteArray();
00395
00396     QByteArray ret;
00397     QByteArray expandedKey = expandKey(key);
00398     QByteArray alignedText(rawText);
00399
00400     //Fill array with padding
00401     alignedText.append(getPadding(rawText.size(), m_blocklen));
00402
00403     switch(m_mode)
00404     {
00405     case ECB:
00406         for(int i=0; i < alignedText.size(); i+= m_blocklen)
00407             ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00408         break;
00409     case CBC: {
00410         QByteArray ivTemp(iv);
00411         for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00412             alignedText.replace(i, m_blocklen, byteXor(alignedText.mid(i, m_blocklen),ivTemp));
00413             ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00414             ivTemp = ret.mid(i, m_blocklen);
00415         }
00416     }
00417         break;
00418     case CFB: {
00419         ret.append(byteXor(alignedText.left(m_blocklen), cipher(expandedKey, iv)));
00420         for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00421             if (i+m_blocklen < alignedText.size())
00422                 ret.append(byteXor(alignedText.mid(i+m_blocklen, m_blocklen),
00423                              cipher(expandedKey, ret.mid(i, m_blocklen))));
00424         }
00425     }
00426         break;
00427     case OFB: {
00428         QByteArray ofbTemp;
00429         ofbTemp.append(cipher(expandedKey, iv));
00430         for (int i=m_blocklen; i < alignedText.size(); i += m_blocklen){
00431             ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00432         }
00433         ret.append(byteXor(alignedText, ofbTemp));
00434     }
00435         break;
00436     default: break;
00437     }
00438     return ret;
00439 }
00440
00441 QByteArray QAESEncryption::decode(const QByteArray &rawText, const QByteArray &key,
      const QByteArray &iv)
00442 {
00443     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00444         return QByteArray();
00445
00446     QByteArray ret;
00447     QByteArray expandedKey = expandKey(key);
00448
00449     switch(m_mode)
00450     {
00451     case ECB:
00452         for(int i=0; i < rawText.size(); i+= m_blocklen)
00453             ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00454         break;
00455     case CBC: {
00456         QByteArray ivTemp(iv);
00457         for(int i=0; i < rawText.size(); i+= m_blocklen){
```

```
00458                         ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00459                         ret.replace(i, m_blocklen, byteXor(ret.mid(i, m_blocklen),ivTemp));
00460                         ivTemp = rawText.mid(i, m_blocklen);
00461                     }
00462               }
00463            break;
00464       case CFB: {
00465               ret.append(byteXor(rawText.mid(0, m_blocklen), cipher(expandedKey, iv)));
00466               for(int i=0; i < rawText.size(); i+= m_blocklen){
00467                   if (i+m_blocklen < rawText.size()) {
00468                       ret.append(byteXor(rawText.mid(i+m_blocklen, m_blocklen),
00469                                     cipher(expandedKey, rawText.mid(i, m_blocklen))));
00470                   }
00471               }
00472           }
00473            break;
00474       case OFB: {
00475           QByteArray ofbTemp;
00476           ofbTemp.append(cipher(expandedKey, iv));
00477           for (int i=m_blocklen; i < rawText.size(); i += m_blocklen){
00478               ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00479           }
00480           ret.append(byteXor(rawText, ofbTemp));
00481       }
00482            break;
00483       default:
00484           //do nothing
00485            break;
00486       }
00487       return ret;
00488 }
00489
00490 QByteArray QAESEncryption::removePadding(const QByteArray &rawText)
00491 {
00492       QByteArray ret(rawText);
00493       switch (m_padding)
00494       {
00495       case Padding::ZERO:
00496           //Works only if the last byte of the decoded array is not zero
00497           while (ret.at(ret.length()-1) == 0x00)
00498               ret.remove(ret.length()-1, 1);
00499            break;
00500       case Padding::PKCS7:
00501           ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00502            break;
00503       case Padding::ISO:
00504           ret.truncate(ret.lastIndexOf(0x80));
00505            break;
00506       default:
00507           //do nothing
00508            break;
00509       }
00510       return ret;
00511 }
```

8.25 qaesencryption.h File Reference

```
#include <QObject>
#include <QByteArray>
```

Include dependency graph for qaesencryption.h:
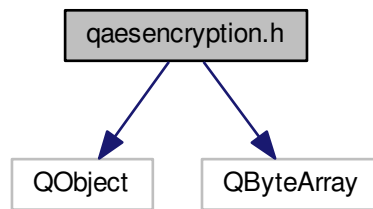
This graph shows which files directly or indirectly include this file:
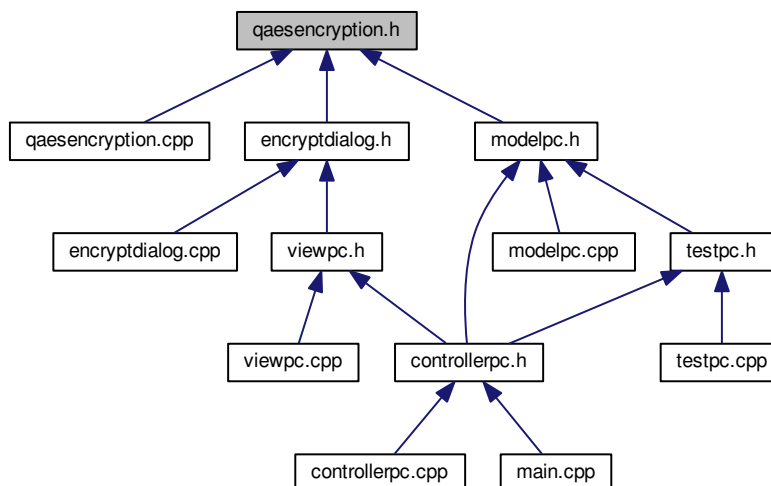
Classes

- class **QAESEncryption**

  The QAESEncryption class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: `https://github.com/bricke/-Qt-AES`.

8.26 qaesencryption.h

```
00001 #ifndef QAESENCRYPTION_H
00002 #define QAESENCRYPTION_H
00003
00004 #include <QObject>
00005 #include <QByteArray>
00006
00014 class QAESEncryption : public QObject
00015 {
```

```
00016      Q_OBJECT
00017 public:
00027     enum Aes {
00028         AES_128,
00029         AES_192,
00030         AES_256
00031     };
00040     enum Mode {
00041         ECB,
00042         CBC,
00043         CFB,
00044         OFB
00045     };
00046
00055     enum Padding {
00056       ZERO,
00057       PKCS7,
00058       ISO
00059     };
00071     static QByteArray Crypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
00072                             const QByteArray &iv = NULL, QAESEncryption::Padding
      padding = QAESEncryption::ISO);
00084     static QByteArray Decrypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
00085                             const QByteArray &iv = NULL,
      QAESEncryption::Padding padding = QAESEncryption::ISO);
00094     static QByteArray ExpandKey(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &key);
00102     static QByteArray RemovePadding(const QByteArray &rawText,
      QAESEncryption::Padding padding);
00103
00104     QAESEncryption(QAESEncryption::Aes level,
      QAESEncryption::Mode mode,
00105                      QAESEncryption::Padding padding =
      QAESEncryption::ISO);
00116     QByteArray encode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv = NULL);
00127     QByteArray decode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv = NULL);
00136     QByteArray removePadding(const QByteArray &rawText);
00145     QByteArray expandKey(const QByteArray &key);
00146
00147 signals:
00148
00149 public slots:
00150
00151 private:
00152     int m_nb;
00153     int m_blocklen;
00154     int m_level;
00155     int m_mode;
00156     int m_nk;
00157     int m_keyLen;
00158     int m_nr;
00159     int m_expandedKey;
00160     int m_padding;
00161     QByteArray* m_state;
00162
00163     struct AES256{
00164         int nk = 8;
00165         int keylen = 32;
00166         int nr = 14;
00167         int expandedKey = 240;
00168     };
00169
00170     struct AES192{
00171         int nk = 6;
00172         int keylen = 24;
00173         int nr = 12;
00174         int expandedKey = 209;
00175     };
00176
00177     struct AES128{
00178         int nk = 4;
00179         int keylen = 16;
00180         int nr = 10;
00181         int expandedKey = 176;
00182     };
00183
00184     quint8 getSBoxValue(quint8 num){return sbox[num];}
00185     quint8 getSBoxInvert(quint8 num){return rsbox[num];}
00186
00187     void addRoundKey(const quint8 round, const QByteArray expKey);
00188     void subBytes();
00189     void shiftRows();
00190     void mixColumns();
00191     void invMixColumns();
00192     void invSubBytes();
```

```
00193      void invShiftRows();
00194      QByteArray getPadding(int currSize, int alignment);
00195      QByteArray cipher(const QByteArray &expKey, const QByteArray &plainText);
00196      QByteArray invCipher(const QByteArray &expKey, const QByteArray &plainText);
00197      QByteArray byteXor(const QByteArray &in, const QByteArray &iv);
00198
00199      const quint8 sbox[256] =    {
00200        //0   1     2     3     4     5     6     7     8     9     A     B     C     D     E     F
00201        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
00202        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
00203        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
00204        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
00205        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
00206        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
00207        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
00208        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
00209        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
00210        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
00211        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
00212        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
00213        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
00214        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
00215        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
00216        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };
00217
00218      const quint8 rsbox[256] =
00219      { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
00220        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
00221        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
00222        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
00223        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
00224        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
00225        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
00226        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
00227        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
00228        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
00229        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
00230        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
00231        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
00232        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
00233        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
00234        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };
00235
00236      // The round constant word array, Rcon[i], contains the values given by
00237      // x to th e power (i-1) being powers of x (x is denoted as {02}) in the field GF(2^8)
00238      // Only the first 14 elements are needed
00239      const quint8 Rcon[256] = {
00240          0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab/*, 0x4d, 0x9a,
00241          0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
00242          0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
00243          0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
00244          0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
00245          0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
00246          0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
00247          0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
00248          0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
00249          0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
00250          0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
00251          0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
00252          0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
00253          0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
00254          0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
00255          0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
      */};
00256 };
00257
00258 #endif // QAESENCRYPTION_H
```

8.27 testpc.cpp File Reference

```
#include "testpc.h"
```

Include dependency graph for testpc.cpp:

8.28 testpc.cpp

```
00001 #include "testpc.h"
00005 TestPC::TestPC()
00006 {
00007     model = new ModelPC();
00008 }
00013 int TestPC::Test()
00014 {
00015     return TestPC().startTest();
00016 }
00028 bool TestPC::test(QByteArray data, QImage rImage, QString expectedOutput, int mode, QString
     key, int bitsUsed)
00029 {
00030     // Error outputs
00031     QString error1, error2;
00032     // Embedding
00033     QImage * retImage = model->encrypt(data, &rImage, mode, key, bitsUsed, &error1);
00034     // De-embedding
00035     QByteArray output = model->decrypt(retImage, key, mode, &error2);
00036
00037     // Success of error outputs
00038     bool er1 = error1 == expectedOutput;
00039     bool er2 = error2 == expectedOutput;
00040     if(expectedOutput == "ok")
00041         return er1 && er2 && data == output;
00042     else
00043         return er1 || er2;
00044 }
00052 int TestPC::startTest()
00053 {
00054     qDebug() << "Testing started...\n";
00055     model = new ModelPC();
00056
00057     // Long text open
00058     QFile file(":/unit_tests/longtext.txt");
00059     if(!file.open(QFile::ReadOnly))
00060         return false;
00061     text = file.readAll();
00062     file.close();
00063
00064     // Big picture open
00065     image = QImage(":/unit_tests/bigpicture.jpg");
00066     if(image.isNull())
00067         return false;
00068
00069     // JSON tests list open
00070     QFile json_file(":/unit_tests/tests.json");
00071     QJsonDocument doc;
00072     if(!json_file.open(QFile::ReadOnly | QFile::Text))
00073         return false;
00074     QByteArray readData = json_file.readAll();
00075     json_file.close();
00076     doc = QJsonDocument::fromJson(readData);
00077     // Testing
00078     return autoTest(doc);
00079 }
00087 bool TestPC::autoTest(QJsonDocument doc)
00088 {
00089     // Opening the tests array
00090     QJsonObject o = doc.object();
00091     QJsonArray arr = o["tests"].toArray();
00092     int sum = 0;
00093
00094     // Info about tests
00095     QString extraText;
00096     for(int i = 0; i < arr.size(); i++) {
00097         // Reading the data
00098         QJsonObject obj = arr[i].toObject();
```

```
00099
00100          QString t = obj["data"].toString();
00101          if(t == "/text/")
00102              t = text;
00103          QByteArray data = t.toUtf8();
00104
00105          QString im = obj["image"].toString();
00106          QImage img(":/unit_tests/" + im);
00107
00108          QString expect = obj["expectation"].toString();
00109
00110          int mode = obj["mode"].toInt();
00111
00112          QString key = obj["key"].toString();
00113
00114          int bitsUsed = obj["bitsUsed"].toInt();
00115
00116          // Testing
00117          bool s = test(data, img, expect, mode, key, bitsUsed);
00118
00119          sum += s;
00120          extraText += "\n * Test #" + QString::number(i + 1) + " " + (s ? "completed." : "failed.");
00121      }
00122      // Writing log
00123      QFile file("tests.log");
00124      bool testsSuc = sum == arr.size();
00125      if(!file.open(QFile::WriteOnly | QFile::Text))
00126          return testsSuc;
00127      QDateTime curTime = QDateTime::currentDateTime();
00128      QString date = curTime.toString("dd.MM.yyyy HH:mm");
00129      QString logtext = "###########################################\n"
00130                        "####Log file created at " + date + "####\n"
00131                        "###########################################\n\n"
00132                        "Status: " + (testsSuc ? "All tests completed" : "Tests failed") + "\n\n"
00133                        "Tests list:\n";
00134      logtext += extraText;
00135      file.write(logtext.toUtf8());
00136      file.close();
00137      // Cleaning up
00138      qDebug() << "Testing completed\n";
00139      delete model;
00140      return !testsSuc;
00141 }
```

8.29 testpc.h File Reference

```
#include <QObject>
#include <modelpc.h>
#include <QFile>
#include <QDebug>
#include <QString>
#include <QImage>
#include <QByteArray>
#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
#include <QDateTime>
```

Include dependency graph for testpc.h:
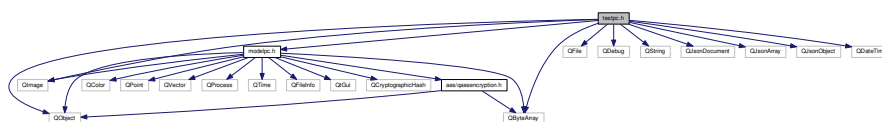
This graph shows which files directly or indirectly include this file:
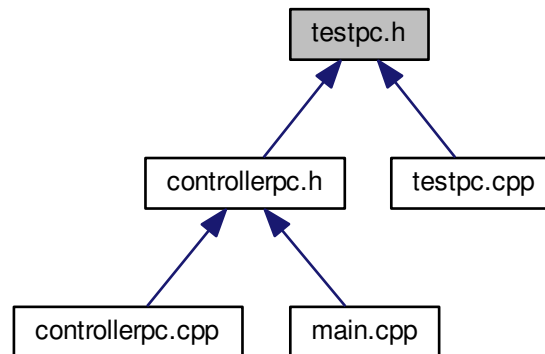
Classes

- class TestPC

  The TestPC class AutoTest for ModelPC Currently used in main.cpp.

8.30 testpc.h

```
00001 #ifndef TESTPC_H
00002 #define TESTPC_H
00003
00004 #include <QObject>
00005 #include <modelpc.h>
00006
00007 #include <QFile>
00008 #include <QDebug>
00009 #include <QString>
00010 #include <QImage>
00011 #include <QByteArray>
00012
00013 #include <QJsonDocument>
00014 #include <QJsonArray>
00015 #include <QJsonObject>
00016
00017 #include <QDateTime>
00022 class TestPC : public QObject
00023 {
00024     Q_OBJECT
00025 public:
00026     TestPC();
00027     static int Test();
00028     // TODO add static Test();
00029 public slots:
00030     int startTest();
00031 protected slots:
00032     bool test(QByteArray data, QImage rImage,
00033             QString expectedOutput = "ok", int mode = 0,
00034             QString key = "", int bitsUsed = 8);
00035 private:
00039     ModelPC * model;
00043     QByteArray text;
00047     QImage image;
00048
00049     bool autoTest(QJsonDocument doc);
00050 };
00051
00052 #endif // TESTPC_H
```

8.31 tests-setup.py File Reference

Namespaces

- tests-setup

Variables

- string tests-setup.filename = 'tests.json'
- tuple tests-setup.raw = open(filename, 'r')
- tuple tests-setup.js = json.load(raw)
- tuple tests-setup.input_data = input()
- list tests-setup.arr = []
- dictionary tests-setup.obj = {'data':data, 'image':image,'expectation':expect,'mode':int(mode),'key':key,'bits-Used':int(bitsUsed)}

8.32 tests-setup.py

```
00001 import json
00002 filename = 'tests.json'
00003
00004 raw = open(filename, 'r')
00005
00006 js = json.load(raw)
00007 print('Existing tests:')
00008 for obj in js['tests']:
00009     print(obj['data'], obj['image'], obj['expectation'], obj['mode'], obj['key'], obj['bitsUsed'], sep='-')
00010
00011 print('------------')
00012 print('Type new tests')
00013
00014 input_data = input()
00015
00016 arr = []
00017 while len(input_data):
00018     data, image, expect, mode, key, bitsUsed = map(str, input_data.split('-'))
00019
00020     obj = {'data':data, 'image':image,'expectation':expect,'mode':int(mode),'key':key,'bitsUsed':int(
bitsUsed)}
00021     arr.append(obj)
00022     input_data = input()
00023
00024 js['tests'] += arr
00025 with open(filename, 'w') as f:
00026     json.dump(js, f, indent=4)
```

8.33 tests.json File Reference

8.34 tests.json

```
00001 {
00002     "tests": [
00003         {
00004             "data": "/text/",
00005             "image": "bigpicture.jpg",
00006             "expectation": "ok",
00007             "mode": 1,
00008             "key": "qwertykey",
00009             "bitsUsed": 8
00010         },
00011         {
00012             "data": "/text/",
00013             "image": "bigpicture.jpg",
00014             "expectation": "ok",
00015             "mode": 1,
00016             "key": "password",
00017             "bitsUsed": 5
```

```
00018         },
00019         {
00020             "data": "/text/",
00021             "image": "bigpicture.jpg",
00022             "expectation": "ok",
00023             "mode": 1,
00024             "key": "wowthatpassword",
00025             "bitsUsed": 1
00026         },
00027         {
00028             "data": "/text/",
00029             "image": "tinypicture.png",
00030             "expectation": "muchdata",
00031             "mode": 1,
00032             "key": "get123",
00033             "bitsUsed": 8
00034         },
00035         {
00036             "data": "",
00037             "image": "bigpicture.jpg",
00038             "expectation": "nodata",
00039             "mode": 1,
00040             "key": "42",
00041             "bitsUsed": 8
00042         },
00043         {
00044             "data": "/text/",
00045             "image": "invalid.jpg",
00046             "expectation": "nullimage",
00047             "mode": 1,
00048             "key": "blog it",
00049             "bitsUsed": 8
00050         },
00051         {
00052             "data": "/text/",
00053             "image": "bigpicture.jpg",
00054             "expectation": "bitsWrong",
00055             "mode": 1,
00056             "key": "turtles are great",
00057             "bitsUsed": 12
00058         },
00059         {
00060             "data": "/text/",
00061             "image": "bigpicture.jpg",
00062             "expectation": "no_key",
00063             "mode": 1,
00064             "key": "",
00065             "bitsUsed": 7
00066         }
00067     ]
00068 }
```

8.35 viewpc.cpp File Reference

```
#include "viewpc.h"
#include "ui_viewpc.h"
```
Include dependency graph for viewpc.cpp:

8.36 viewpc.cpp

```
00001 #include "viewpc.h"
00002 #include "ui_viewpc.h"
00003
00004 ViewPC::ViewPC(QWidget *parent) :
```

```
00005     QMainWindow(parent),
00006     ui(new Ui::ViewPC)
00007 {
00008     ui->setupUi(this);
00009
00010     progressDialogClosed = true;
00011
00012     // Alerts dictionary setup
00013     QFile file(":/config/ErrorsDict.json");
00014     if(!file.open(QFile::ReadOnly | QFile::Text)) {
00015         alert("Cannot open config file!");
00016         return;
00017     }
00018     QByteArray readData = file.readAll();
00019     file.close();
00020
00021     QJsonParseError error;
00022     QJsonDocument doc = QJsonDocument::fromJson(readData, &error);
00023     errorsDict = doc.object();
00024 }
00028 ViewPC::~ViewPC()
00029 {
00030     delete ui;
00031 }
00032
00033 void ViewPC::on_encryptMode_clicked()
00034 {
00035     // Encrypt radio button clicked
00036     setEncryptMode(true);
00037 }
00038
00039 void ViewPC::on_decryptMode_clicked()
00040 {
00041     // Decrypt radio button clicked
00042     setEncryptMode(false);
00043 }
00047 void ViewPC::on_fileButton_clicked()
00048 {
00049     // Opening QFileDialog depending on isEncrypt
00050     if(isEncrypt)
00051         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.txt", tr("Text
      files (*.txt);;All Files (*)"));
00052     else
00053         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.png", tr("PNG
      files (*.png);;All Files (*)"));
00054     // Display the file name
00055     ui->fileLabel->setText(inputFileName.isEmpty() ? "File not chosen" : inputFileName);
00056 }
00069 void ViewPC::on_startButton_clicked()
00070 {
00071     if(isEncrypt)
00072     {
00073         // Getting the data
00074         QString text = ui->text->toPlainText();
00075         QByteArray data;
00076         if(text.isEmpty()) {
00077             if(inputFileName.isEmpty()) {
00078                 alert("No input file or text was not given. Cannot continue!", true);
00079                 return;
00080             }
00081             // Opening the file
00082             QFile file(inputFileName);
00083             if (!file.open(QIODevice::ReadOnly))
00084             {
00085                 alert("Cannot open file. Cannot continue!", true);
00086                 return;
00087             }
00088             // Check the data size
00089             auto size = file.size();
00090             if(size > qPow(2, 24)) {
00091                 alert("Your file is too big, our systems can handle it, but it requires a lot of time.
      We decline.", true);
00092                 file.close();
00093                 return;
00094             }
00095             data = file.readAll();
00096             file.close();
00097         }
00098         else
00099             data = text.toUtf8();
00100         // Select image via EncryptDialog
00101         EncryptDialog * dialog = new EncryptDialog(data);
00102         dialog->exec();
00103         if(!dialog->success)
00104             return;
00105
00106         // Get the data
```

```
00107            QByteArray encr_data = dialog->compr_data;
00108
00109            // Save the hash
00110            QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
00111            encr_data = hash + encr_data;
00112            // TODO do the mode thing
00113            emit encrypt(encr_data, &dialog->image, 1, dialog->bitsUsed);
00114        }
00115    else
00116    {
00117            // Get the filename of the image
00118            if(!ui->text->toPlainText().isEmpty())
00119                alert("Obviously, the text browser isn't supported for decryption, use File Dialog
    instead.");
00120            if(inputFileName.isEmpty()) {
00121                alert("File not selected. Cannot continue!", true);
00122                return;
00123            }
00124            QByteArray key = requestKey().toUtf8();
00125            if(key.isEmpty())
00126                return;
00127            QImage * res_image = new QImage(inputFileName);
00128            emit decrypt(res_image, key, 1);
00129        }
00130 }
00136 void ViewPC::alert(QString message, bool isWarning)
00137 {
00138    // Get message
00139    if(errorsDict.contains(message))
00140        message = errorsDict[message].toString();
00141    // Create message box
00142    QMessageBox box;
00143    if(isWarning)
00144        box.setIcon(QMessageBox::Warning);
00145    else
00146        box.setIcon(QMessageBox::Information);
00147    box.setText(message);
00148    box.setWindowIcon(QIcon(":/icons/mail.png"));
00149    box.setWindowTitle("Message");
00150    box.exec();
00151 }
00157 void ViewPC::saveData(QByteArray Edata)
00158 {
00159    // Save data using QFileDialog
00160    QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save File"),
00161                              "/untitled.txt",
00162                              tr("Text(*.txt);;All files (*)"));
00163    QFile writeFile(outputFileName);
00164    if (!writeFile.open(QIODevice::WriteOnly))
00165    {
00166        alert("Cannot access file path. Cannot continue!", true);
00167        return;
00168    }
00169    writeFile.write(Edata);
00170    writeFile.close();
00171    alert("Decryption completed!");
00172 }
00178 void ViewPC::saveImage(QImage * image)
00179 {
00180    // Save image using QFileDialog
00181    QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save Image"),
00182                              "/untitled.png",
00183                              tr("Images(*.png)"));
00184    if(!image->save(outputFileName)) {
00185        alert("Cannot save file. Unable to continue!", true);
00186        return;
00187    }
00188    alert("Encryption completed!");
00189 }
00196 void ViewPC::setProgress(int val)
00197 {
00198    if(val < 0) {
00199        // Create dialog
00200        dialog = new QProgressDialog("Cryption in progress.", "Cancel", 0, 100);
00201        connect(dialog, SIGNAL(canceled()), this, SLOT(abortCircuit()));
00202        progressDialogClosed = false;
00203        dialog->setWindowTitle("Processing");
00204        dialog->setWindowIcon(QIcon(":/icons/loading.png"));
00205        dialog->show();
00206    }
00207    else if(val >= 100 && !progressDialogClosed) {
00208        // Close dialog
00209        dialog->setValue(100);
00210        QThread::msleep(25);
00211        dialog->close();
00212        dialog->reset();
00213        progressDialogClosed = true;
```

```
00214      }
00215      // Update the progress
00216      else if(!progressDialogClosed)
00217          dialog->setValue(val);
00218  }
00222  void ViewPC::abortCircuit()
00223  {
00224      // Set the flag
00225      progressDialogClosed = true;
00226      // Close the dialog
00227      dialog->close();
00228      dialog->reset();
00229      emit abortModel();
00230  }
00235  void ViewPC::setEncryptMode(bool encr)
00236  {
00237      ui->text->setEnabled(encr);
00238      isEncrypt = encr;
00239      ui->startButton->setText(encr ? "Continue configuration" : "Start decryption");
00240      ui->enLabel1->setText(encr ? "Type in the text for encryption:" : "Text input isn't supported in
       decryption mode");
00241      ui->enLabel1->setEnabled(encr);
00242      ui->enLabel2->setText(encr ? "Or use the file dialog to choose a file:" : "Choose a file for
       decryption:");
00243  }
00248  void ViewPC::setVersion(QString version)
00249  {
00250      // Version setup
00251      versionString = version;
00252  }
00257  QString ViewPC::requestKey()
00258  {
00259      bool ok;
00260      QString text = QInputDialog::getText(this, tr("QInputDialog::getText()"),
00261                                           tr("Enter the keyphrase:"), QLineEdit::Normal,
00262                                           QDir::home().dirName(), &ok);
00263      if(text.isEmpty() && ok) {
00264          alert("Key is empty!", true);
00265          return QString();
00266      }
00267      return ok ? text : QString();
00268  }
00269
00270  QByteArray ViewPC::bytes(long long n)
00271  {
00272      return QByteArray::fromHex(QByteArray::number(n, 16));
00273  }
00277  void ViewPC::on_actionAbout_triggered()
00278  {
00279      AboutPC about;
00280      about.setVersion(versionString);
00281      about.exec();
00282  }
00283
00287  void ViewPC::on_actionHelp_triggered()
00288  {
00289      QUrl docLink("https://alexkovrigin.me/PictureCrypt");
00290      QDesktopServices::openUrl(docLink);
00291  }
00292
00293  void ViewPC::on_actionJPHS_path_triggered()
00294  {
00295      QString dir = QFileDialog::getExistingDirectory(this, tr("Open JPHS folder"),
00296                                                      "/home",
00297                                                      QFileDialog::ShowDirsOnly
00298                                                      | QFileDialog::DontResolveSymlinks);
00299      emit setJPHSDir(dir);
00300  }
00301
00302  void ViewPC::on_actionRun_tests_triggered()
00303  {
00304      emit runTests();
00305  }
```

8.37 viewpc.h File Reference

```
#include <QMainWindow>
```

```
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QImage>
#include <QByteArray>
#include <QVector>
#include <QThread>
#include <QDesktopServices>
#include <QInputDialog>
#include <QtMath>
#include <encryptdialog.h>
#include <QProgressDialog>
#include <aboutpc.h>
#include <QJsonDocument>
#include <QJsonObject>
```
Include dependency graph for viewpc.h:
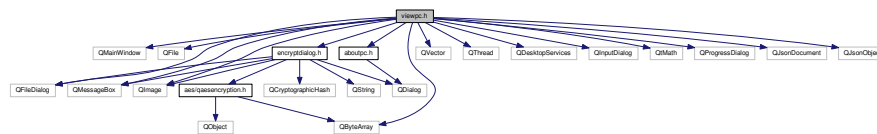
This graph shows which files directly or indirectly include this file:
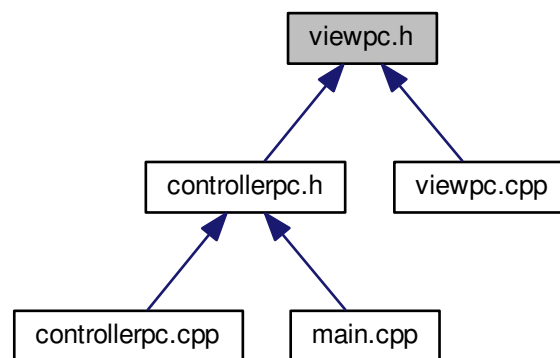
Classes

- class ViewPC

    The ViewPC class View layer of the app. Controls EncryptDialog and ProgressDialog.

Namespaces

- Ui

8.37.1 Detailed Description

Header of ViewPC class

See Also

> ControllerPC, ModelPC, ViewPC

Definition in file viewpc.h.

8.38 viewpc.h

```
00001 #ifndef VIEWPC_H
00002 #define VIEWPC_H
00003
00004 #include <QMainWindow>
00005 #include <QFile>
00006 #include <QFileDialog>
00007 #include <QMessageBox>
00008 #include <QImage>
00009 #include <QByteArray>
00010 #include <QVector>
00011 #include <QThread>
00012 #include <QDesktopServices>
00013 #include <QInputDialog>
00014 #include <QtMath>
00015
00016 #include <encryptdialog.h>
00017 #include <QProgressDialog>
00018 #include <aboutpc.h>
00019
00020 #include <QJsonDocument>
00021 #include <QJsonObject>
00022
00023 namespace Ui {
00024 class ViewPC;
00025 }
00035 class ViewPC : public QMainWindow
00036 {
00037     Q_OBJECT
00038
00039 public:
00040     explicit ViewPC(QWidget *parent = nullptr);
00041     ~ViewPC();
00042 private slots:
00043     void on_encryptMode_clicked();
00044
00045     void on_decryptMode_clicked();
00046
00047     void on_actionJPHS_path_triggered();
00048
00049     void on_actionRun_tests_triggered();
00050
00051 protected slots:
00052     void on_fileButton_clicked();
00053
00054     void on_startButton_clicked();
00055
00056     void on_actionAbout_triggered();
00057
00058     void on_actionHelp_triggered();
00059 public slots:
00060     void alert(QString message, bool isWarning = false);
00061     void saveData(QByteArray Edata);
00062     void saveImage(QImage *image);
00063     void setProgress(int val);
00064     void abortCircuit();
00065     void setEncryptMode(bool encr);
00066     void setVersion(QString version);
00067 signals:
00075     void encrypt(QByteArray data, QImage * image, int mode, int bitsUsed);
00082     void decrypt(QImage * _image, QString key, int mode);
00086     void abortModel();
00091     void setJPHSDir(QString dir);
00095     void runTests();
00096 public:
00101     QProgressDialog * dialog;
00106     bool progressDialogClosed;
00107     QJsonObject errorsDict;
00108 protected:
```

```
00109     QString requestKey();
00110 private:
00111     Ui::ViewPC *ui;
00112     bool isEncrypt;
00113     QString inputFileName;
00114     QByteArray bytes(long long n);
00115     QString versionString;
00116 };
00117
00118 #endif // VIEWPC_H
```

Index