PictureCrypt

1.4.0

Generated by Doxygen 1.8.11

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.

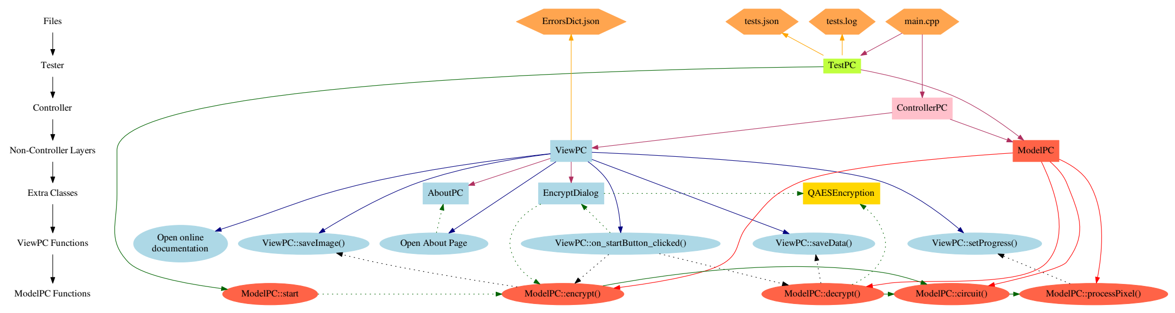
Project is done via MVC Pattern. View and Model layers are totally isolated and run on different threads.

Code from controller.cpp

```
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.

```
#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)

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'
- raw = open(filename, 'r')
- data = json.load(raw)
- input_data = input()
- key
- value
- f
- indent

6.1.1 Variable Documentation

6.1.1.1 ErrorsDictSetup.data = json.load(raw)

Definition at line 6 of file ErrorsDictSetup.py.

6.1.1.2 ErrorsDictSetup.f

Definition at line 22 of file ErrorsDictSetup.py.

6.1.1.3 string ErrorsDictSetup.filename = 'ErrorsDict.json'

Definition at line 2 of file ErrorsDictSetup.py.

6.1.1.4 ErrorsDictSetup.indent

Definition at line 22 of file ErrorsDictSetup.py.

6.1.1.5 ErrorsDictSetup.input_data = input()

Definition at line 14 of file ErrorsDictSetup.py.

6.1.1.6 ErrorsDictSetup.key

Definition at line 17 of file ErrorsDictSetup.py.

6.1.1.7 ErrorsDictSetup.raw = open(filename, 'r')

Definition at line 4 of file ErrorsDictSetup.py.

6.1.1.8 ErrorsDictSetup.value

Definition at line 17 of file ErrorsDictSetup.py.

6.2 Ui Namespace Reference

Chapter 7

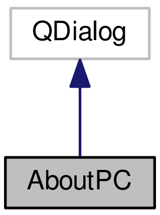
Class Documentation

7.1 AboutPC Class Reference

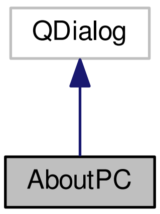
The AboutPC class The About Page dialog.

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

Inheritance diagram for AboutPC:



Collaboration diagram for 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*)

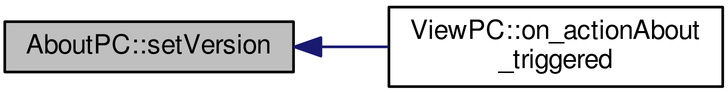
AboutPC::setVersion Function to set the version display.

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:

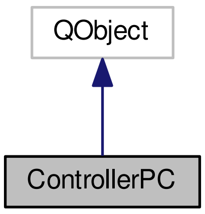
- aboutpc.h
- aboutpc.cpp

7.2 ControllerPC Class Reference

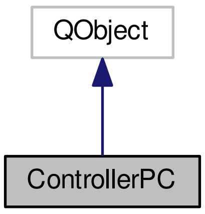
The ControllerPC class Controller class, which controls View and Model layers.

```
#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 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 20 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.

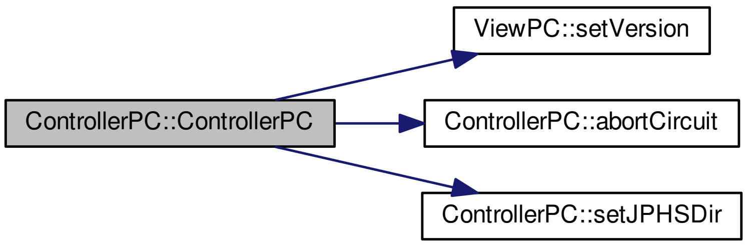
Controller class

Note

Version of the app is specified here.

Definition at line 9 of file controllerpc.cpp.

Here is the call graph for this function:



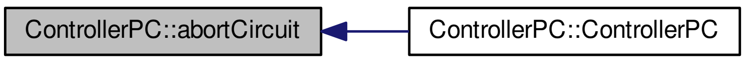
7.2.3 Member Function Documentation

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::setJPHSDir (QString *dir*) `[slot]`

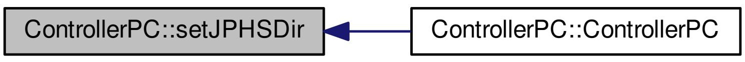
ControllerPC::setJPHSDir Sets JPHS default dir.

Parameters

dir	Directory

Definition at line 45 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 28 of file controllerpc.h.

7.2.4.2 QString ControllerPC::versionString

versionString Version of the app as QString.

Definition at line 32 of file controllerpc.h.

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

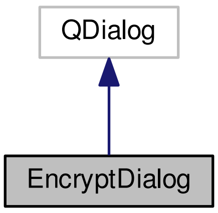
- controllerpc.h
- controllerpc.cpp

7.3 EncryptDialog Class Reference

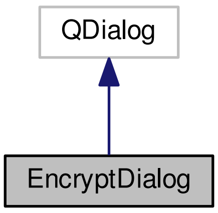
The EncryptDialog class Class to get the image and key to store secret info.

```
#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]

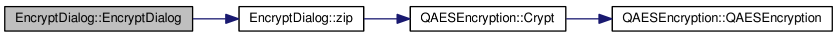
EncryptDialog::EncryptDialog Constructor of the class. Input data is saved here and some variables are set here.

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

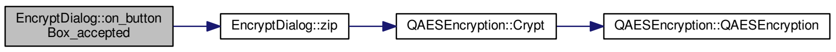
Definition at line 107 of file encryptdialog.cpp.

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

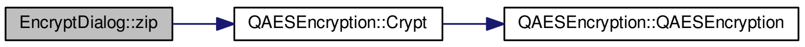
Returns Compressed data.

See also

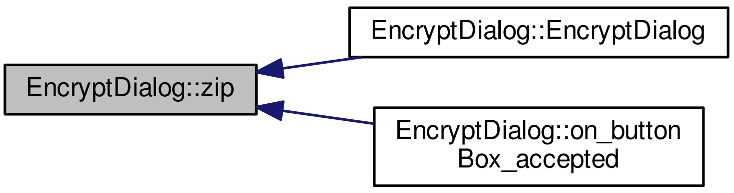
ModelPC::unzip

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:

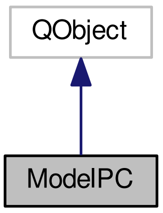
- encryptdialog.h
- encryptdialog.cpp

7.4 ModelPC Class Reference

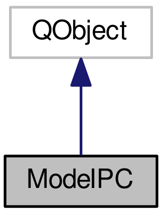
The ModelPC class Model Layer of the app. Main class that does the work of PictureCrypt logic Controled by ControllerPC.

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

Inheritance diagram for ModelPC:



Collaboration diagram for ModelPC:



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+.
- void proccessPixelsv1_4 (QImage ∗image, QByteArray ∗data, QByteArray key, bool isEncrypt, QVector< QPair< QPoint, QPair< int, int > > > ∗were, long long size=-1)

  ModelPC::proccessPixelsv1_4 Hides (or retrieves) data to/from pixels.
- 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 33 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 38 of file modelpc.h.

7.4.3 Constructor & Destructor Documentation

7.4.3.1 ModelPC::ModelPC ( )

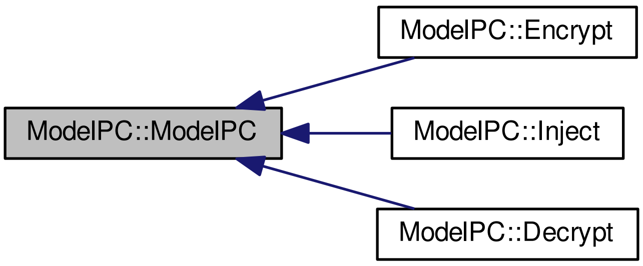
ModelPC::ModelPC Constructor Unit tests are run here.

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

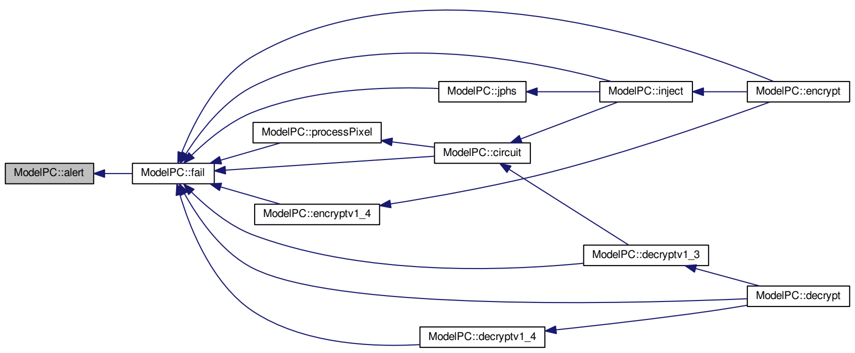
message	Message to be transmitted.
isWarning	Flag if message is critical.

See also

ViewPC::alert

Definition at line 940 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.

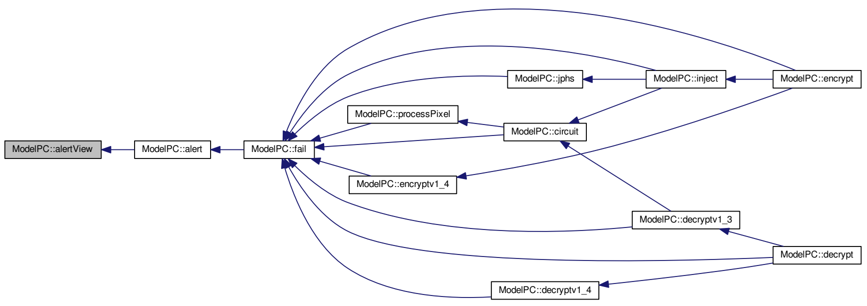
Parameters

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

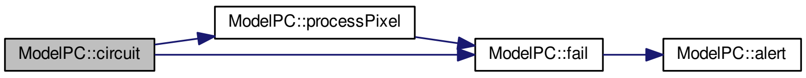
image	Image to be processed.
data	Data to be processed.
countBytes	Number of bytes to be read or written.

See also

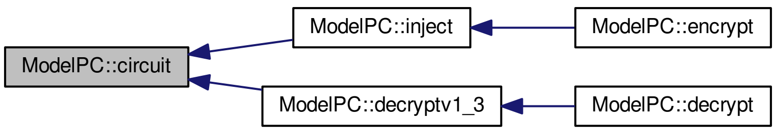
ModelPC::processPixel

Definition at line 359 of file modelpc.cpp.

Here is the call graph for this function:



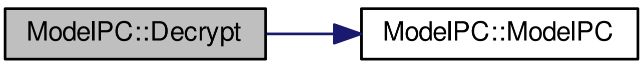
Here is the caller graph for this function:



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

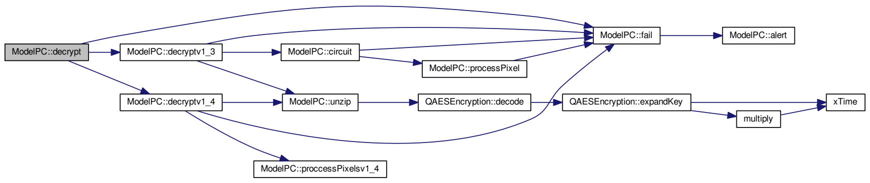
Returns decrypted data

See also

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

Definition at line 212 of file modelpc.cpp.

Here is the call 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

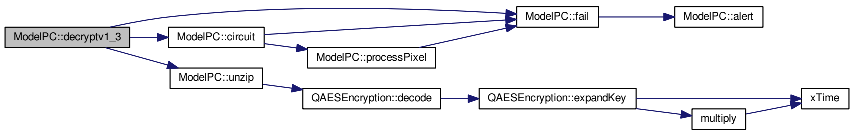
image	Image with data
key	Key

Returns

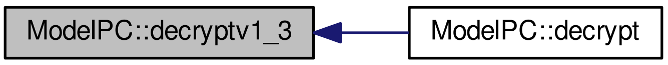
Returns obtained data

Definition at line 777 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

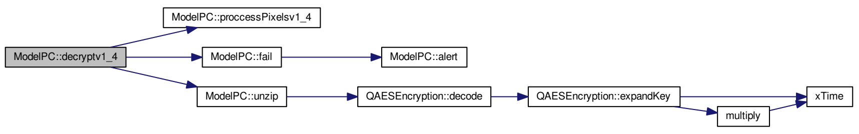
image	Image with data
key	Key

Returns

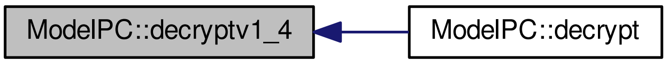
Returns obtained data

Definition at line 602 of file modelpc.cpp.

Here is the call graph for this function:



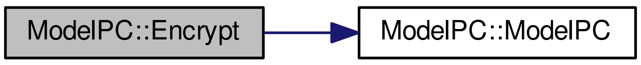
Here is the caller graph for this function:



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

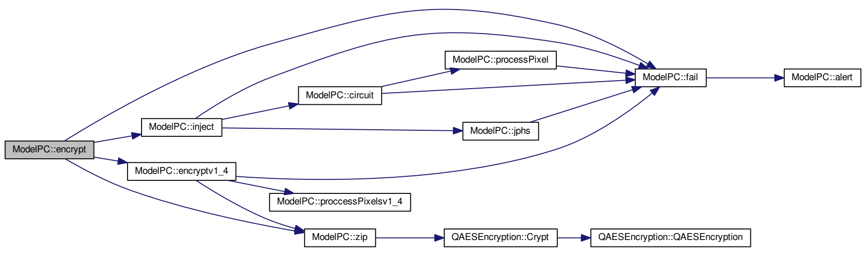
Returns image with embedded data

See also

ModelPC::inject

Definition at line 51 of file modelpc.cpp.

Here is the call graph for this function:



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

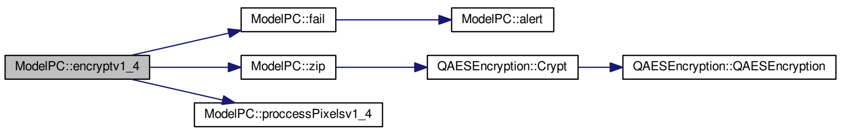
ModelPC::encryptv1_4 Encrypts and injects data to image used in v1.4+.

Parameters

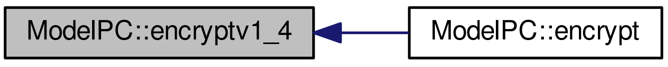
image	Image for injecting
data	Data for embedding

Definition at line 560 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]`

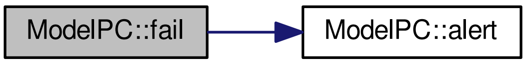
ModelPC::fail Slot to stop execution of cryption.

Parameters

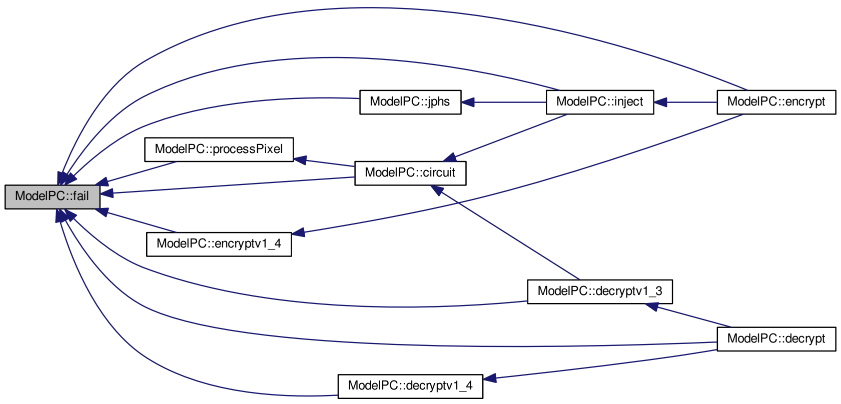
message	Message for user

Definition at line 283 of file modelpc.cpp.

Here is the call graph for this function:



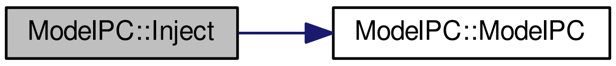
Here is the caller graph for this function:



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:



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

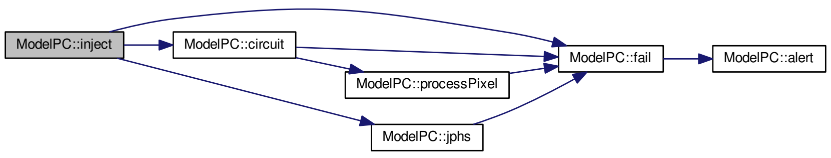
Returns image with embedded data.

See also

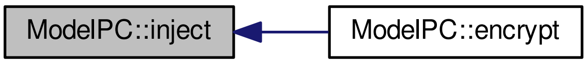
ViewPC::on_startButton_clicked, ModelPC::decrypt, ModelPC::circuit, ModelPC::start

Definition at line 139 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]

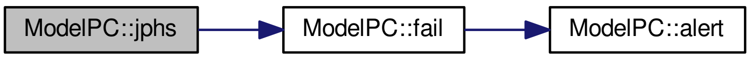
ModelPC::jphs JPHS function to use jphide and jpseek (currently under development)

Parameters

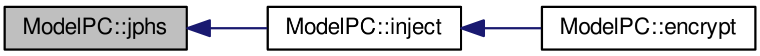
image	Image for embedding
data	Data

Definition at line 298 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::proccessPixelsv1_4 (QImage ∗ *image,* QByteArray ∗ *data,* QByteArray *key,* bool *isEncrypt,* QVector< QPair< QPoint, QPair< int, int > > > ∗ *were,* long long *size =* −1) [protected]

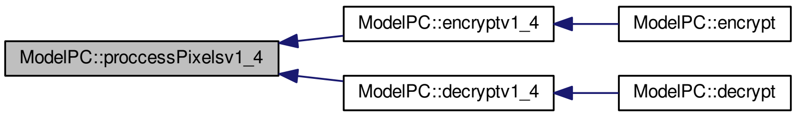
ModelPC::proccessPixelsv1_4 Hides (or retrieves) data to/from pixels.

Parameters

image	Original image
data	Data to write (Pointer to empty QByteArray if decrypting)
key	Key
isEncrypt	Mode of Cryption (true -> encryption, false -> decryption)
were	Were vector for visited pixels
size	Size of reading data, unneeded if writing

Definition at line 663 of file modelpc.cpp.

Here is the caller graph for this function:



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

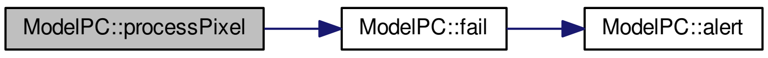
ModelPC::processPixel Processes every pixel. Reads its contains or writes data.

Parameters

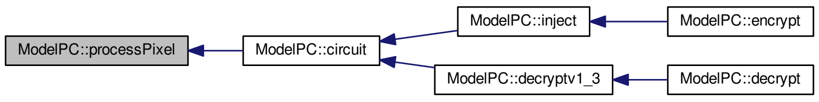
pos	Position of pixel
were	Vector array containing pixels, that were already processed.
isEncrypt	Mode of operation. If true encryption operations will continue, else the decryption ones.

Definition at line 500 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



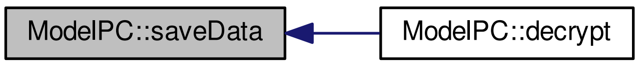
7.4.4.17 void ModelPC::saveData (QByteArray *data*) [signal]

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:



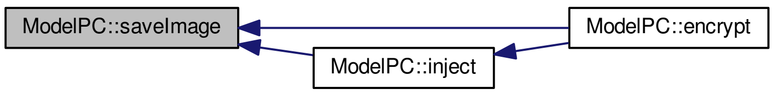
7.4.4.18 void ModelPC::saveImage (QImage ∗ *image*) [signal]

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:



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

setProgress Signal to be called to set progress of ProgressDialog.

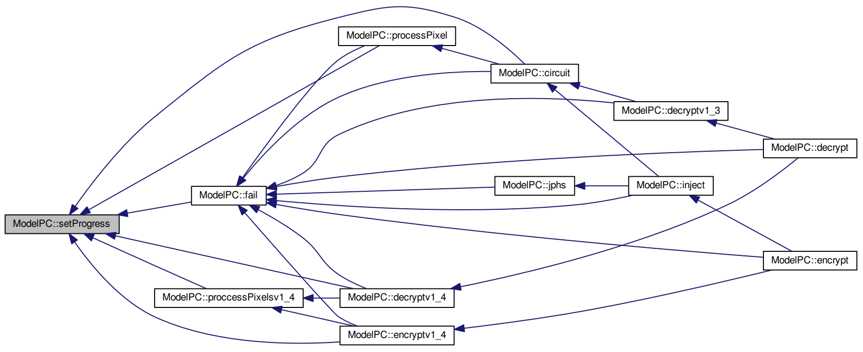
Parameters

val	Value to be set.

See also

ViewPC::setProgress

Here is the caller graph for this function:



7.4.4.20 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

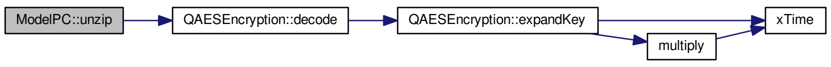
Returns data

See also

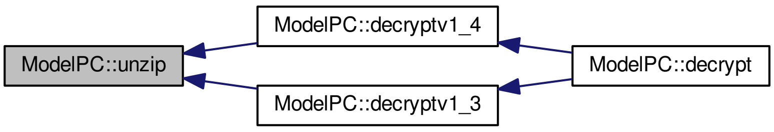
EncryptDialog::zip, ModelPC::decrypt, ModelPC::zip

Definition at line 879 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.4.21 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

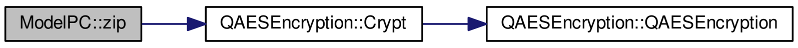
Returns decrypted data

See also

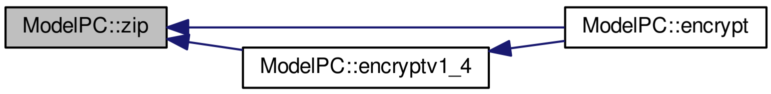
ModelPC::start, ModelPC::inject, ModelPC::unzip

Definition at line 896 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 94 of file modelpc.h.

7.4.5.2 QString∗ ModelPC::error [protected]

error Current error

Definition at line 108 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 82 of file modelpc.h.

7.4.5.4 long ModelPC::version

version Version of the class

Definition at line 86 of file modelpc.h.

7.4.5.5 QString ModelPC::versionString

versionString Version as string

Definition at line 90 of file modelpc.h.

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

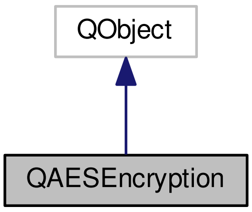
- modelpc.h
- modelpc.cpp

7.5 QAESEncryption Class Reference

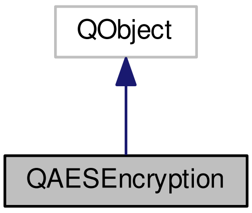
The QAESEncryption class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: https://github.↩ com/bricke/Qt-AES.

```
#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=QA←ESEncryption::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=QAE←SEncryption::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

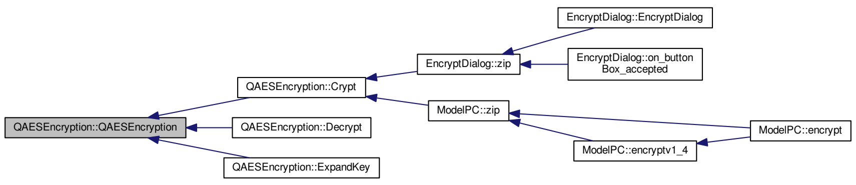
Definition at line 55 of file qaesencryption.h.

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:


EncryptDialog::EncryptDialog
EncryptDialog::zip
EncryptDialog::on_button Box_accepted
QAESEncryption::Crypt
ModelPC::zip
QAESEncryption::QAESEncryption
QAESEncryption::Decrypt
ModelPC::encrypt
ModelPC::encryptv1_4
QAESEncryption::ExpandKey

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

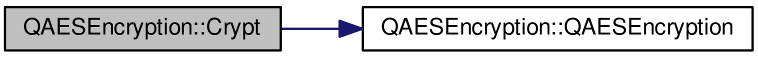
Returns encrypted data

See also

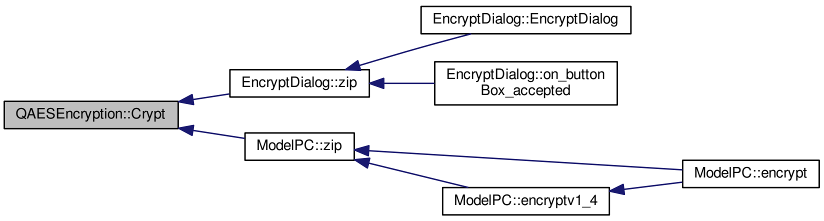
QAESEncryption::encode, QAESEncryption::Decrypt

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

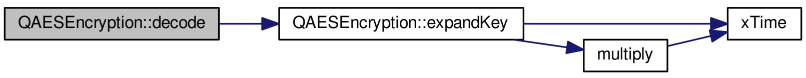
Returns decoded data

See also

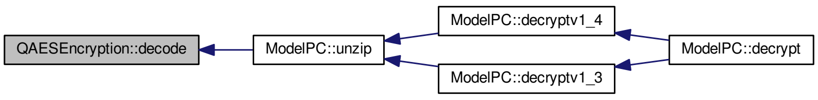
QAESEncryption::Decrypt, QAESEncryption::encode

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

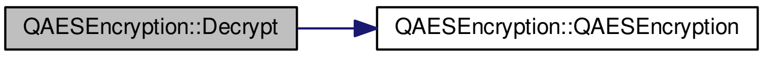
Returns Decrypted data

See also

QAESEncryption::decode, QAESEncryption::Crypt

Definition at line 12 of file qaesencryption.cpp.

Here is the call graph for this function:



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

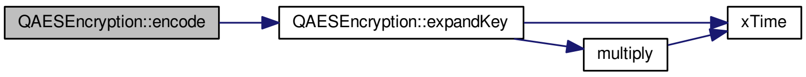
Returns encoded data

See also

QAESEncryption::Crypt, QAESEncryption::decode

Definition at line 391 of file qaesencryption.cpp.

Here is the call graph for this function:



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

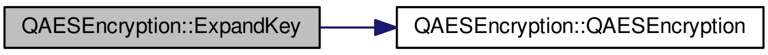
Returns expanded key (I guess)

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

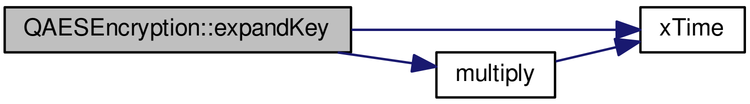
Returns expanded key (I guess)

See also

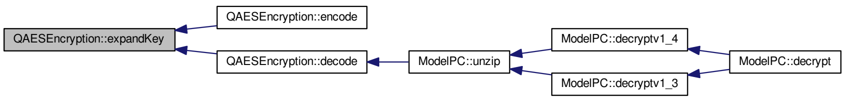
QAESEncryption::ExpandKey

Definition at line 132 of file qaesencryption.cpp.

Here is the call graph for this function:



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.

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:

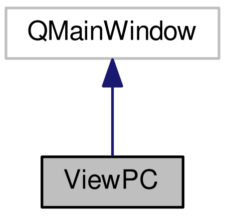
- qaesencryption.h
- qaesencryption.cpp

7.6 ViewPC Class Reference

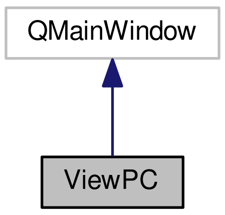
The ViewPC class View layer of the app. Controls EncryptDialog and ProgressDialog.

```
#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, QString key)

  encrypt Signal calling ModelPC::encrypt
- void inject (QByteArray data, QImage ∗image, int mode, int bitsUsed)

  inject Signal calling ModelPC::inject
- 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

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

  errorsDict Json object for errors dictionary

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.

7.6.1 Detailed Description

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

See also

ControllerPC, ModelPC, EncryptDialog

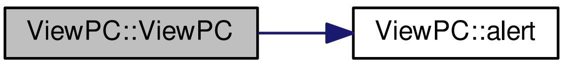
Definition at line 35 of file viewpc.h.

7.6.2 Constructor & Destructor Documentation

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

Definition at line 4 of file viewpc.cpp.

Here is the call graph for this function:

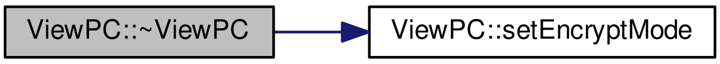


7.6.2.2 ViewPC::∼ViewPC ( )

ViewPC::∼ViewPC Simple destructor for this layer.

Definition at line 29 of file viewpc.cpp.

Here is the call graph for this function:



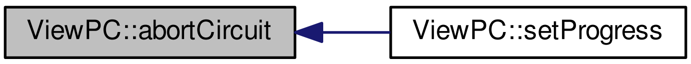
7.6.3 Member Function Documentation

7.6.3.1 void ViewPC::abortCircuit ( ) [slot]

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

Definition at line 228 of file viewpc.cpp.

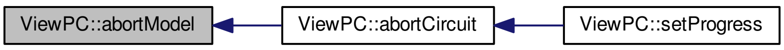
Here is the caller graph for this function:



7.6.3.2 void ViewPC::abortModel ( ) [signal]

abortModel Signal calling to stop ModelPC::circuit

Here is the caller graph for this function:



7.6.3.3 void ViewPC::alert (QString *message*, bool *isWarning* = false) [slot]

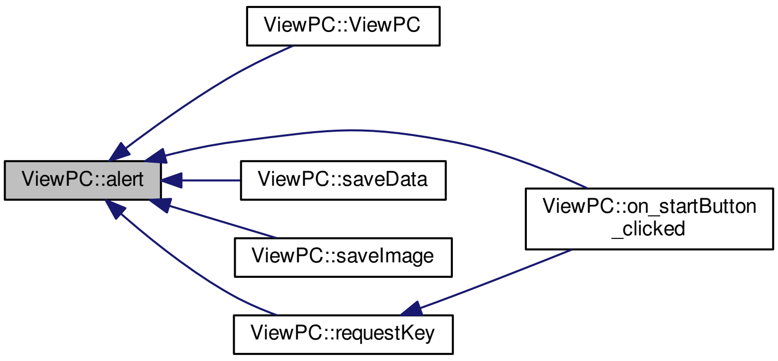
ViewPC::alert Slot to create QMessageBox with message.

Parameters

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

Definition at line 142 of file viewpc.cpp.

Here is the caller graph for this function:



7.6.3.4 void ViewPC::decrypt (QImage ∗ *_image,* QString *key,* int *mode*) [signal]

decrypt Signal calling ModelPC::decrypt

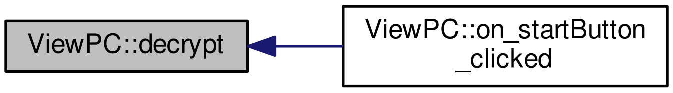
Parameters

_image	Image for decryption
key	encryption key
mode	Mode of decryption

See also

ModelPC::decrypt, ModelPC::CryptMode

Here is the caller graph for this function:



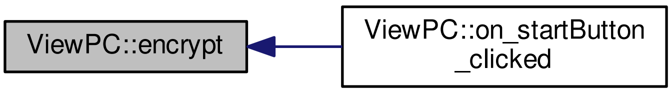
7.6.3.5 void ViewPC::encrypt (QByteArray *data*, QImage ∗ *image*, int *mode*, QString *key*) [signal]

encrypt Signal calling ModelPC::encrypt

Parameters

data	Data to write
image	Image to be encrypted into
mode	Mode of encryption
key	Key of encryption

Here is the caller graph for this function:



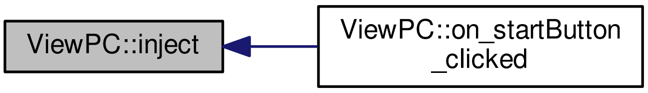
7.6.3.6 void ViewPC::inject (QByteArray *data*, QImage ∗ *image*, int *mode*, int *bitsUsed*) [signal]

inject Signal calling ModelPC::inject

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:

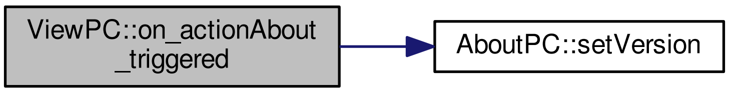


7.6.3.7 void ViewPC::on_actionAbout_triggered ( ) [protected], [slot]

ViewPC::on_actionAbout_triggered Opens about page.

Definition at line 285 of file viewpc.cpp.

Here is the call graph for this function:



7.6.3.8 void ViewPC::on_actionHelp_triggered ( ) [protected], [slot]

ViewPC::on_actionHelp_triggered Opens online documentation.

Definition at line 295 of file viewpc.cpp.

7.6.3.9 void ViewPC::on_fileButton_clicked ( ) [protected], [slot]

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

Definition at line 48 of file viewpc.cpp.

7.6.3.10 void ViewPC::on_startButton_clicked ( ) [protected], [slot]

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

7.6.4 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

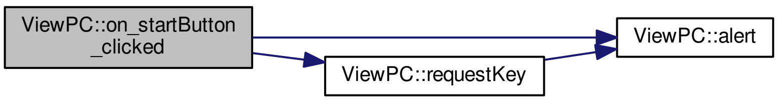
Then the EncryptDialog opens and image and key is selected. Then the ViewPC::encrypt signal is called to start ModelPC::encrypt

7.6.5 Decrypting

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

Definition at line 70 of file viewpc.cpp.

Here is the call graph for this function:



7.6.5.1 QString ViewPC::requestKey ( ) [protected]

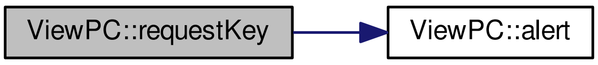
ViewPC::requestKey Request keyphrase from user using InputDialog.

Returns

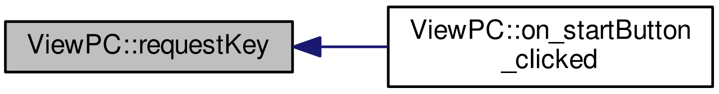
Returns keyphrase

Definition at line 265 of file viewpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Parameters

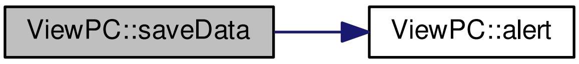
Edata	Encrypted data to be saved.

See also

ModelPC::encrypt

Definition at line 163 of file viewpc.cpp.

Here is the call graph for this function:



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

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

Parameters

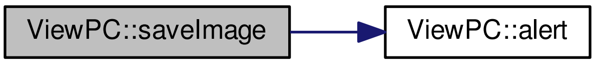
image	Image to be saved.

See also

ModelPC::decrypt

Definition at line 184 of file viewpc.cpp.

Here is the call graph for this function:



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

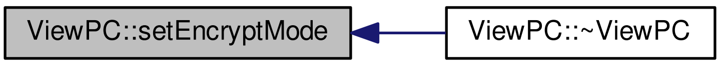
ViewPC::setEncryptMode Set the encrpt mode (ViewPC::isEncrypt)

Parameters

encr	= isEncrypt, true if encrypting, false if decrypting

Definition at line 241 of file viewpc.cpp.

Here is the caller graph for this function:



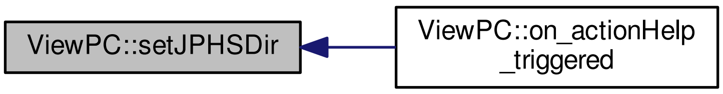
7.6.5.5 void ViewPC::setJPHSDir (QString *dir*) `[signal]`

setJPHSPath Sets the default JPHS directory

Parameters

dir	Directory

Here is the caller graph for this function:



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

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

Parameters

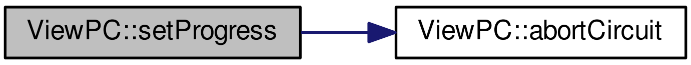
val	New value of the dialog. If -1, creates ProgressDialog, if 101 closes the dialog.

See also

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

Definition at line 202 of file viewpc.cpp.

Here is the call graph for this function:



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

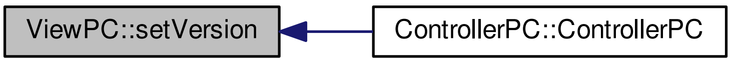
ViewPC::setVersion Set the version of the app from ControllerPC.

Parameters

version	Version as QString

Definition at line 256 of file viewpc.cpp.

Here is the caller graph for this function:



7.6.6 Member Data Documentation

7.6.6.1 QProgressDialog∗ ViewPC::dialog

dialog ProgressDialog used.

See also

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

Definition at line 108 of file viewpc.h.

7.6.6.2 QJsonObject ViewPC::errorsDict

errorsDict Json object for errors dictionary

Definition at line 117 of file viewpc.h.

7.6.6.3 bool ViewPC::progressDialogClosed

progressDialogClosed Flag, if dialog is closed.

See also

ViewPC::abortCircuit, ViewPC::setProgress

Definition at line 113 of file viewpc.h.

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

- viewpc.h
- viewpc.cpp

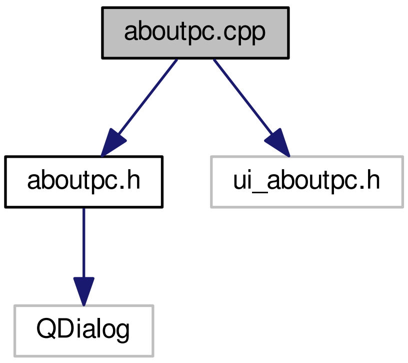
Chapter 8

File Documentation

8.1 aboutpc.cpp File Reference

```
#include "aboutpc.h"
#include "ui_aboutpc.h"
```

Include dependency graph for aboutpc.cpp:



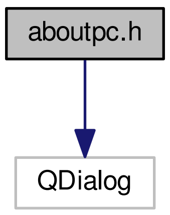
8.2 aboutpc.cpp

```
#include "aboutpc.h"
#include "ui_aboutpc.h"

AboutPC::AboutPC(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::AboutPC)
{
    ui->setupUi(this);
}

AboutPC::~AboutPC()
{
    delete ui;
}
void AboutPC::setVersion(QString version)
{
    ui->versionLabel->setText("Version " + version);
}
```

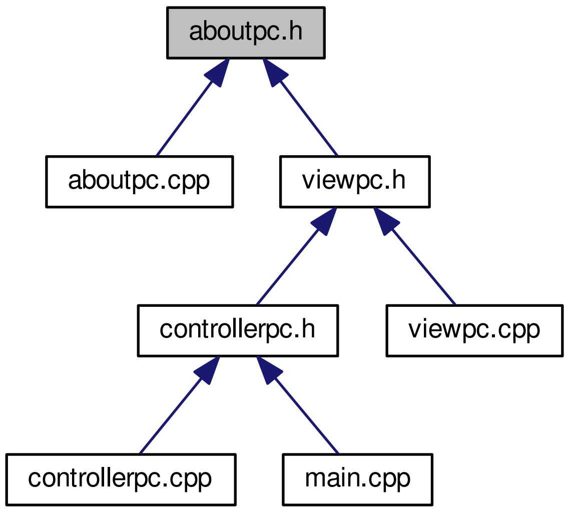
8.3 aboutpc.h File Reference

`#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

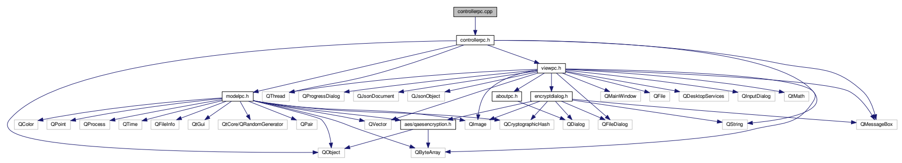
8.4 aboutpc.h

```
#ifndef ABOUTPC_H
#define ABOUTPC_H

#include <QDialog>

namespace Ui {
class AboutPC;
}
class AboutPC : public QDialog
{
    Q_OBJECT

public:
    explicit AboutPC(QWidget *parent = 0);
    ~AboutPC();
    void setVersion(QString version);

private:
    Ui::AboutPC *ui;
};

#endif // ABOUTPC_H
```

8.5 controllerpc.cpp File Reference

`#include "controllerpc.h"`

Include dependency graph for controllerpc.cpp:



8.6 controllerpc.cpp

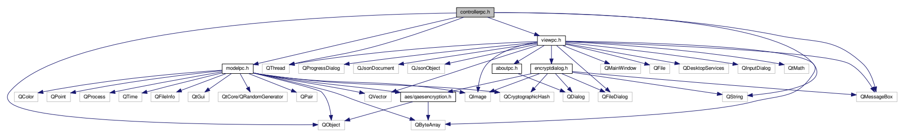
```
#include "controllerpc.h"

ControllerPC::ControllerPC()
{
    // Layer creation
    view = new ViewPC();
    model = new ModelPC();
    QThread * modelThread = new QThread();
    model->moveToThread(modelThread);
    modelThread->start();

    view->setVersion(model->versionString);
    view->show();

    // Layers Connection
    connect(view, SIGNAL(encrypt(QByteArray, QImage*, int, QString)), model, SLOT(encrypt(QByteArray, QImage*, int, QString)));
    connect(view, SIGNAL(inject(QByteArray,QImage*,int, int)), model, SLOT(inject(QByteArray,QImage*, int, int)));
    connect(view, SIGNAL(decrypt(QImage*,QString,int)), model, SLOT(decrypt(QImage*, QString, int)));
    connect(view, SIGNAL(abortModel()), this, SLOT(abortCircuit()));
    connect(view, SIGNAL(setJPHSDir(QString)), this, SLOT(setJPHSDir(QString)));

    connect(model, SIGNAL(alertView(QString,bool)), view, SLOT(alert(QString,bool)));
    connect(model, SIGNAL(saveData(QByteArray)), view, SLOT(saveData(QByteArray)));
    connect(model, SIGNAL(saveImage(QImage*)), view, SLOT(saveImage(QImage*)));
    connect(model, SIGNAL(setProgress(int)), view, SLOT(setProgress(int)));
}
void ControllerPC::abortCircuit()
```

```
{
    model->success = false;
}
void ControllerPC::setJPHSDir(QString dir)
{
    model->defaultJPHSDir = dir;
}
```

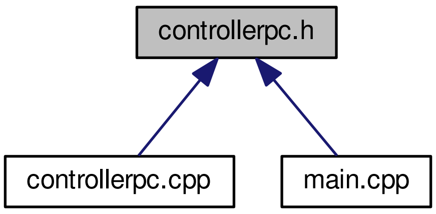
8.7 controllerpc.h File Reference

```
#include <QObject>
#include <QString>
#include <QThread>
#include <QMessageBox>
#include <modelpc.h>
#include <viewpc.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.

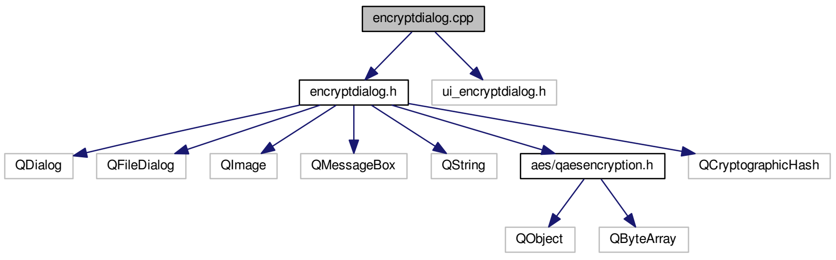
8.8 controllerpc.h

```
#ifndef CONTROLLERPC_H
#define CONTROLLERPC_H

#include <QObject>
#include <QString>
#include <QThread>
#include <QMessageBox>

#include <modelpc.h>
#include <viewpc.h>
class ControllerPC : public QObject
{
    Q_OBJECT
public:
    ControllerPC();
    long int version;
    QString versionString;
public slots:
    void abortCircuit();
    void setJPHSDir(QString dir);
private:
    ViewPC * view;
    ModelPC * model;
};

#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

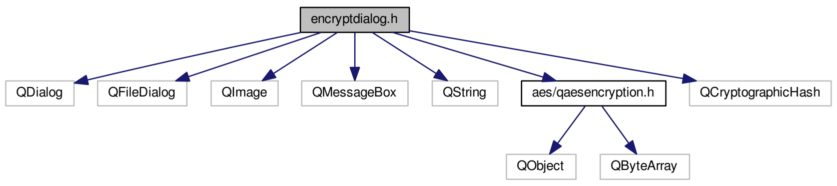
```
#include "encryptdialog.h"
#include "ui_encryptdialog.h"
EncryptDialog::EncryptDialog(QByteArray _data, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::EncryptDialog)
{
    ui->setupUi(this);
    data = _data;
    success = false;
    // UI setup
    ui->totalBytes->setText(QString::number(data.size()));
    key = "";
    compr_data = zip();
    long long int compr_data_size = compr_data.size();
    ui->zippedBytes->setText(QString::number(compr_data_size));
    goodPercentage = false;
    bitsUsed = 8;
}

EncryptDialog::~EncryptDialog()
{
    delete ui;
}

void EncryptDialog::alert(QString text)
{
    QMessageBox t;
    t.setWindowTitle("Message");
    t.setIcon(QMessageBox::Warning);
    t.setWindowIcon(QIcon(":/mail.png"));
    t.setText(text);
    t.exec();
}
QByteArray EncryptDialog::zip()
{
    // Zip
    QByteArray c_data = qCompress(data, 9);
    // Encryption
    QByteArray hashKey = QCryptographicHash::hash(key.toUtf8(), QCryptographicHash::Sha256);
    return QAESEncryption::Crypt(QAESEncryption::AES_256, QAESEncryption::ECB, c_data, hashKey);
}
void EncryptDialog::on_fileButton_clicked()
{
    // Selet file
    inputFileName = QFileDialog::getOpenFileName(this, tr("Open File"), "/", tr("Images (*.png *.xpm *.jpg *.jpeg)"));
    ui->fileLabel->setText(inputFileName);
    // Open image
    QImage img(inputFileName);
    image = img;
    // Get size
    size = img.width() * img.height();
    // UI setup
    long long int compr_data_size = compr_data.size();
    ui->zippedBytes->setText(QString::number(compr_data_size));
    if(inputFileName.isEmpty()) {
        ui->percentage->setText("");
        return;
    }
    double perc = (compr_data_size + 14) * 100 / (size * 3) * bitsUsed / 8;
    ui->percentage->setText(QString::number(perc) + "%");
    goodPercentage = perc < 70;
}
void EncryptDialog::on_buttonBox_accepted()
{
    if(!goodPercentage) {
        alert("Your encoding percentage is over 70% which is a bit ambiguous :(");
        success = false;
        return;
    }
    // Final zip
    key = ui->keyLine->text();
    compr_data = zip();
    success = true;
    close();
}
void EncryptDialog::on_buttonBox_rejected()
{
    success = false;
    close();
}
void EncryptDialog::on_bitsSlider_valueChanged(int value)
{
    bitsUsed = value;
    ui->bitsUsedLbl->setText(QString::number(value));
    if(ui->percentage->text().isEmpty())
        return;
    double perc = (compr_data.size() + 14) * 100 / (size * 3) * 8 / bitsUsed;
    ui->percentage->setText(QString::number(perc) + "%");
}
```

8.11 encryptdialog.h File Reference

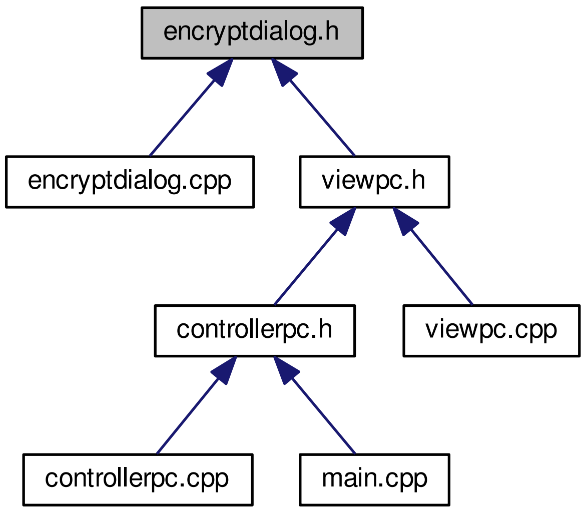
```
#include <QDialog>
```

```
#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

```
#ifndef ENCRYPTDIALOG_H
#define ENCRYPTDIALOG_H

#include <QDialog>
#include <QFileDialog>
#include <QImage>
#include <QMessageBox>
#include <QString>

#include <aes/qaesencryption.h>
#include <QCryptographicHash>

namespace Ui {
class EncryptDialog;
}
class EncryptDialog : public QDialog
{
    Q_OBJECT

public:
    explicit EncryptDialog(QByteArray _data, QWidget *parent = 0);
    ~EncryptDialog();

public slots:
    void on_fileButton_clicked();

    void on_buttonBox_accepted();

    void on_buttonBox_rejected();

    void on_bitsSlider_valueChanged(int value);

public:
    QByteArray data;
    bool success;
    QByteArray compr_data;
    QString inputFileName;
    long long int size;
    QString key;
    bool goodPercentage;
    int val;
    int bitsUsed;
    QImage image;
    QByteArray zip();
private:
    Ui::EncryptDialog *ui;
    void alert(QString text);
};

#endif // ENCRYPTDIALOG_H
```

8.13 ErrorsDict.json File Reference

8.14 ErrorsDict.json

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

8.15 ErrorsDictSetup.py File Reference

Namespaces

- ErrorsDictSetup

Variables

- string ErrorsDictSetup.filename = 'ErrorsDict.json'
- ErrorsDictSetup.raw = open(filename, 'r')
- ErrorsDictSetup.data = json.load(raw)
- ErrorsDictSetup.input_data = input()
- ErrorsDictSetup.key
- ErrorsDictSetup.value
- ErrorsDictSetup.f
- ErrorsDictSetup.indent

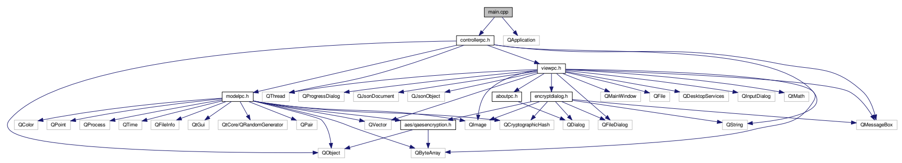
8.16 ErrorsDictSetup.py

```
import json
filename = 'ErrorsDict.json'

raw = open(filename, 'r')

data = json.load(raw)
print('Existing data:')
for key, value in data.items():
    print(key, value)

print('------------')
print('Type new data')

input_data = input()

while len(input_data):
    key, value = map(str, input_data.split('-'))
    data[key] = value
    input_data = input()

with open(filename, 'w') as f:
    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.

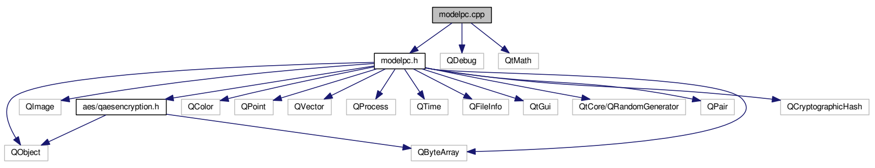
8.18 main.cpp

```
#include "controllerpc.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    ControllerPC w;

    return a.exec();
}
```

8.19 modelpc.cpp File Reference

```
#include "modelpc.h"
#include <QDebug>
#include <QtMath>
```

Include dependency graph for modelpc.cpp:



8.20 modelpc.cpp

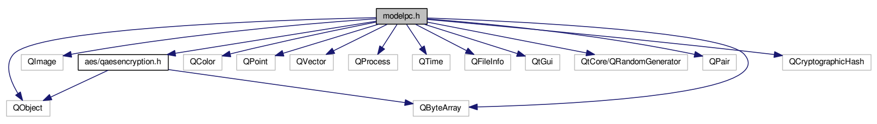
```
#include "modelpc.h"
#include <QDebug>
#include <QtMath>
ModelPC::ModelPC()
{
    // Version control
    versionString = "1.4.0.dev-alpha.4";

    auto ver = versionString.split(".");
    version = ver[0].toInt() * qPow(2, 16) + ver[1].toInt() * qPow(2, 8) + ver[2].toInt();

    ver_byte = bytes(ver[0].toInt()) +
            bytes(ver[1].toInt()) +
            bytes(ver[2].toInt());
    // Random seed
    qsrand(randSeed());
}

QImage *ModelPC::Encrypt(QByteArray data, QImage *image, int _mode, QString
key, int _bitsUsed, QString *_error)
{
    return ModelPC().encrypt(data, image, _mode, key, _bitsUsed, _error);
}

QImage *ModelPC::Inject(QByteArray encr_data, QImage *image, int _mode, int _bitsUsed,
QString *_error)
{
    return ModelPC().inject(encr_data, image, _mode, _bitsUsed, _error);
}

QByteArray ModelPC::Decrypt(QImage *image, QString key, int _mode, QString *_error)
{
    return ModelPC().decrypt(image, key, _mode, _error);
}
QImage * ModelPC::encrypt(QByteArray data, QImage * image, int _mode, QString
key, int _bitsUsed, QString *_error)
{
    success = true;
    CryptMode mode = CryptMode(_mode);
    // Error management
    if(_error == nullptr)
        _error = new QString();
    *_error = "ok";
    error = _error;

    if(data == nullptr || data.isEmpty()) {
        fail("nodata");
        return nullptr;
    }
    if(data.size() > pow(2, 24)) {
        fail("muchdata");
        return nullptr;
    }
    if(image == nullptr || image->isNull()) {
        fail("nullimage");
        return nullptr;
    }
    if(image->width() * image->height() > pow(10, 9)) {
        fail("bigimage");
        return nullptr;
    }
    if(_bitsUsed < 1 || _bitsUsed > 8) {
        fail("bitsWrong");
        return nullptr;
    }
    if(key == nullptr || key.isEmpty()) {
        fail("no_key");
        return nullptr;
    }
    else if(key.size() > 255) {
        fail("bigkey");
        return nullptr;
    }
    if(mode == CryptMode::NotDefined) {
        fail("undefined_mode");
        return nullptr;
    }
    long long usedBytes = data.size() + 14 + key.size();
    long long size = image->width() * image->height();
    if(usedBytes * 100 / (size * 3) * 8 / _bitsUsed > 70) {
        fail("bigdata");
        return nullptr;
    }

    switch(mode)
    {
        case v1_3:
        {
            QByteArray zipped_data = zip(data, key.toUtf8());
            QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
            QByteArray encr_data = hash + zipped_data;
            if(*error == "ok")
                return inject(encr_data, image, _mode, _bitsUsed, error);
            else
                return nullptr;
            break;
        }
        case v1_4:
            bitsUsed = _bitsUsed;
            encryptv1_4(image, data, key);
            emit saveImage(image);
            return image;
        break;
        case jphs_mode:
            // TODO add jphs
            return nullptr;
        break;
        default:
            fail("wrongmode");
            return nullptr;
    }
}

QImage * ModelPC::inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed,
QString *_error)
{
    success = true;
    CryptMode mode = CryptMode(_mode);
    // Error management
    if(_error == nullptr)
        _error = new QString();
    *_error = "ok";
    error = _error;

    bitsUsed = _bitsUsed;
    if(encr_data == nullptr || encr_data.isEmpty()) {
        fail("nodata");
        return nullptr;
    }
    if(encr_data.size() > pow(2, 24)) {
        fail("muchdata");
        return nullptr;
    }
    if(image == nullptr || image->isNull()) {
        fail("nullimage");
        return nullptr;
    }
    if(image->width() * image->height() > pow(10, 9)) {
        fail("bigimage");
        return nullptr;
    }
    if(_bitsUsed < 1 || _bitsUsed > 8) {
        fail("bitsWrong");
        return nullptr;
    }
    if(mode == CryptMode::NotDefined) {
        fail("undefined_mode");
        return nullptr;
    }

    encr_data = ver_byte + encr_data;
    long long int countBytes = encr_data.size();
    switch(mode)
    {
    case v1_3:
        circuit(image, &encr_data, countBytes);
        break;
    case jphs_mode:
        jphs(image, &encr_data);
        break;
    case v1_4:
        fail("inject-v1.4");
        return nullptr;
        break;
    default:
        fail("wrongmode");
        return nullptr;
    }

    // Saving
    if(success) {
        emit saveImage(image);
        return image;
    }
    else
        return nullptr;
}
QByteArray ModelPC::decrypt(QImage * image, QString key, int _mode, QString *_error)
{
    success = true;
    CryptMode mode = CryptMode(_mode);
    // Error management
    if(_error == nullptr)
        _error = new QString();
    *_error = "ok";
    error = _error;
    if(image == nullptr || image->isNull()) {
        fail("nullimage");
        return nullptr;
    }
    if(image->width() * image->height() > pow(10, 9)) {
        fail("bigimage");
        return nullptr;
    }
    if(key == nullptr || key.isEmpty()) {
        fail("no_key");
        return nullptr;
    }
    QByteArray result;

    switch (mode) {
    case v1_3:
        result = decryptv1_3(image, key);
    break;
    case v1_4:
        result = decryptv1_4(image, key);
    break;
    case jphs_mode:
        // TODO add jphs support
    break;
    case NotDefined:
        isTry = true;

        // v1_3
        result = decryptv1_3(new QImage(*image), key);
        if(success) {
            isTry = false;
            break;
        }
        success = true;

        // v1_4
        result = decryptv1_4(image, key);
        if(success) {
            isTry = false;
            break;
        }
        success = true;

        // TODO add jphs support

        isTry = false;
        fail("all_modes_fail");
        return nullptr;
    break;
    default:
        // For invalid modes
        fail("wrongmode");
        return nullptr;
    }
    if(*error == "ok")
        emit saveData(result);
    return result;
}
void ModelPC::fail(QString message)
{
    success = false;
    if(!isTry) {
        *error = message;
        alert(message, true);
        emit setProgress(101);
    }
    qDebug() << "[Debug] !!! fail() - " << message;
}
void ModelPC::jphs(QImage *image, QByteArray *data)
{
    // Under Development
    return;

    // Dead code

    success = true;
    bool isEncrypt = !data->isEmpty();
    QString targetEXE = defaultJPHSDir + (isEncrypt ? "/jphide.exe" : "/jpseek.exe");
    if(!fileExists(targetEXE))
    {
        fail("nojphs");
        return;
    }

    QString randomFileName = defaultJPHSDir + "/";
    qsrand(randSeed());
    for(int i = 0; i < 10; i++)
        randomFileName.append(97 + qrand() % 25);
    image->save(randomFileName + ".jpg");
    if(isEncrypt) {
        QFile file(randomFileName + ".pc");
        if(!file.open(QFile::WriteOnly)) {
            fail("save_file_fail");
            return;
        }
        file.write(*data);
        file.close();

        QStringList args;
        args << (randomFileName + ".jpg") << (randomFileName + "_out.jpg") << (randomFileName + ".pc");
        QProcess prog(this);
        prog.start(targetEXE, args);
        prog.waitForStarted();
        prog.write("test\n");
        prog.waitForBytesWritten();
        prog.write("test\n");
        prog.waitForBytesWritten();
        prog.waitForReadyRead();
        QByteArray bytes = prog.readAll();
        prog.waitForFinished();
        //QByteArray readData = prog.readAll();
        prog.close();
        // Cleaning - Deleting temp files

    }
    else {

    }

}

void ModelPC::circuit(QImage *image, QByteArray *data, long long countBytes)
{
    // Some flags and creation of the ProgressDialog
    success = true;
    emit setProgress(-1);
    bool isEncrypt = !data->isEmpty();

    // Image setup
    int w = image->width();
    int h = image->height();

    // Visited pixels array
    QVector <QPoint> were;
    were.push_back(QPoint(0, 0));
    were.push_back(QPoint(0, h - 1));
    were.push_back(QPoint(w - 1, 0));
    were.push_back(QPoint(w - 1, h - 1));

    long long int offset = 0;

    // Pre-start Cleaning
    circuitData = data;
    circuitImage = image;
    circuitCountBytes = countBytes;
    cur = 0;
    bitsBuffer.clear();

    // Writing Top-Left to Bottom-Left
    for(int i = 1; i < h - 1 && mustGoOn(isEncrypt); i++) {
        QPoint pos(0, i);
        processPixel(pos, &were, isEncrypt);
    }
    // Writing Bottom-Right to Top-Right
    if(mustGoOn(isEncrypt))
    {
        for(int i = h - 2; i >= 1 && mustGoOn(isEncrypt); i--){
            QPoint pos(w - 1, i);
            processPixel(pos, &were, isEncrypt);
        }
    }
    // Main cycle
    // Strong is considered as actual corner pixel and weak as pixel near it like (1, 0) or (0, 1)
    while(mustGoOn(isEncrypt))
    {
        // Strong Top-Right to Strong Bottom-Right
        for(int i = offset; i < h - offset && mustGoOn(isEncrypt); i++){
            QPoint pos(w - offset - 2, i);
            processPixel(pos, &were, isEncrypt);
        }
        // Strong Top-Left to Weak Top-Right
        for(int i = offset + 1; i < w - offset - 2 && mustGoOn(isEncrypt); i++){
            QPoint pos(i, offset);
            processPixel(pos, &were, isEncrypt);
        }
        // Weak Bottom-Right to Weak Bottom-Left
        for(int i = w - 3 - offset; i >= offset + 2 && mustGoOn(isEncrypt); i--){
            QPoint pos(i, h - offset - 1);
            processPixel(pos, &were, isEncrypt);
        }
        // Weak Top-Left to Strong Bottom-Left
        for(int i = offset + 1; i < h - offset && mustGoOn(isEncrypt); i++){
            QPoint pos(offset + 1, i);
            processPixel(pos, &were, isEncrypt);
        }
        offset++;
    }
    // Extra writing
    if(!success)
        return;
    if(isEncrypt)
    {
        // Getting past colors
        QColor colUL = image->pixelColor(0, 0).toRgb();
        QColor colUR = image->pixelColor(w - 1, 0).toRgb();
        QColor colDL = image->pixelColor(0, h - 1).toRgb();
        QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
        int red = 0;
        int green = 0;
        int blue = 0;

        // Writing Upper Left
        red = (colUL.red() & 224) + (countBytes >> 19);
        green = (colUL.green() & 224) + (countBytes >> 14) % 32;
        blue = (colUL.blue() & 224) + (countBytes >> 9) % 32;
        image->setPixelColor(0, 0, QColor(red, green, blue));

        // Writing Upper Right
        red = (colUR.red() & 224) + (countBytes >> 4) % 32;
        green = (colUR.green() & 224) + ((countBytes % 16) << 1) + 1;
        blue = (colUR.blue() & 224) + 9;
        image->setPixelColor(w - 1, 0, QColor(red, green, blue));

        // Getting extra bytes if left
        while(cur < countBytes)
            push(mod(circuitData->at(cur++)), 8);
        if(bitsBuffer.size() > 20) {
            fail("bitsBufferFail");
            return;
        }
        // Getting extra data as long.
        long extraData = pop(-2);

        // Writing Down Left
        red = (colDL.red() & 224) + (extraData >> 15);
        green = (colDL.green() & 224) + (extraData >> 10) % 32;
        blue = (colDL.blue() & 224) + (extraData >> 5) % 32;
        image->setPixelColor(0, h - 1, QColor(red, green, blue));

        // Writing Down Right
        red = (colDR.red() & 224) + extraData % 32;
        green = (colDR.green() & 224);
        blue = (colDR.blue() & 224) + ((bitsUsed - 1) << 2) + 2;
        image->setPixelColor(w - 1, h - 1, QColor(red, green, blue));
    }
    else
    {
        // Read the past pixels
        QColor colDL = image->pixelColor(0, h - 1).toRgb();
        QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();

        // Read extra data
        long extraData = ((colDL.red() % 32) << 15) + ((colDL.green() % 32) << 10);
        extraData += ((colDL.blue() % 32) << 5) + colDR.red() % 32;

        // Add extra data to the bitsBuffer
        push(extraData, (countBytes - cur) * 8 - bitsBuffer.size());

        // Move bits from bitsBuffer to the QByteArray
        while(!bitsBuffer.isEmpty())
            data->append(pop(8));
    }
    emit setProgress(101);
}

void ModelPC::processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt)
{
    if(!success)
        return;
    // Check if point was already visited
    if(were->contains(pos)){
        fail("point_visited_twice");
        return;
    }
    else
        were->push_back(pos);
    if(isEncrypt)
    {
        // Make sure that there are enough bits in bitsBuffer to write
        while(bitsBuffer.size() < 3 * bitsUsed)
            push(mod(circuitData->at(cur++)), 8);
        // Read past contains
        QColor pixelColor = circuitImage->pixelColor(pos);
        int red = pixelColor.red();
        int green = pixelColor.green();
        int blue = pixelColor.blue();

        // Write new data in last bitsUsed pixels
        red += pop() - red % (int) qPow(2, bitsUsed);
        green += pop() - green % (int) qPow(2, bitsUsed);
        blue += pop() - blue % (int) qPow(2, bitsUsed);

        circuitImage->setPixelColor(pos, QColor(red, green, blue));
    }
    else
    {
        QColor read_color = circuitImage->pixelColor(pos).toRgb();
        // Reading the pixel
        int red = read_color.red();
        int green = read_color.green();
        int blue = read_color.blue();

        // Reading the last bitsUsed pixels
        red %= (int) qPow(2, bitsUsed);
        green %= (int) qPow(2, bitsUsed);
        blue %= (int) qPow(2, bitsUsed);

        // Getting the data in the bitsBuffer.
        push(red);
        push(green);
        push(blue);

        // Getting data to QByteArray
        while(bitsBuffer.size() >= 8) {
            circuitData->append(pop(8));
            cur++;
        }
    }
    emit setProgress(100 * cur / circuitCountBytes);
}
void ModelPC::encryptv1_4(QImage *image, QByteArray data, QString key)
{
    if(data.size() + 98 > image->height() * image->width() * 3) {
        fail("bigdata");
        return;
    }
    QTime st = QTime::currentTime();
    QByteArray rand_master = GetRandomBytes(32);
    QByteArray pass = QCryptographicHash::hash(key.toUtf8() + rand_master + QByteArray("hi"), QCryptographicHash::Sha3_384);
    QByteArray noise = GetRandomBytes(data.size() / 10 + 32);
    QByteArray bytes_key = GetRandomBytes(32);
    QByteArray pass_rand = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_512);
    QByteArray zipped = zip(data, pass_rand);
    QByteArray heavy_data = zipped + noise;

    QByteArray verification = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_256);
    QByteArray given_key = bytes_key.left(30);
    QByteArray heavy_data_size;
    // heavy_data_size is always 4 bytes as max for heavy_data is: 2^24 * 11/10 + 32 ~ 1.8 * 10^7 < 2^32
    long long raw_size = zipped.size();
    for(int i = 0; i < 4; i++) {
        int ch = raw_size % 256;
        raw_size >>= 8;
        heavy_data_size.push_front(ch);
    }
    QByteArray mid_data = verification + given_key + rand_master + heavy_data_size;
    // mid_data.size() = 32 + 30 + 32 + 4 = 98
    QVector <QPair<QPoint, QPair<int, int>>> *were = new QVector <QPair<QPoint, QPair<int, int>>>();
    emit setProgress(-1);
    proccessPixelsv1_4(image, &mid_data, key.toUtf8(), true, were);
    proccessPixelsv1_4(image, &heavy_data, pass_rand, true, were);
    emit setProgress(101);
    QTime final = QTime::currentTime();
    qDebug() << "[Debug] Finished encrypting in " << st.msecsTo(final) << " msecs.";
}

QByteArray ModelPC::decryptv1_4(QImage *image, QString key)
{
    QTime st = QTime::currentTime();
    QByteArray mid_data, heavy_data;
    QVector <QPair<QPoint, QPair<int, int>>> *were = new QVector <QPair<QPoint, QPair<int, int>>>();
    emit setProgress(-1);
    proccessPixelsv1_4(image, &mid_data, key.toUtf8(), false, were, 98);
    QByteArray verification = mid_data.left(32);
    QByteArray given_key = mid_data.mid(32, 30);
    QByteArray rand_master = mid_data.mid(62, 32);
    QByteArray heavy_data_size = mid_data.right(4);

    QByteArray pass = QCryptographicHash::hash(key.toUtf8() + rand_master + QByteArray("hi"), QCryptographicHash::Sha3_384);

    // Guessing
    emit setProgress(0);
    QByteArray bytes_key;
    for(long long i = 0; i < pow(2, 16); i++) {
        QByteArray guess_part;
        long long g = i;
        for(int q = 0; q < 2; q++) {
                int ch = g % 256;
                g >>= 8;
                guess_part.push_front(ch);
            }
        emit setProgress(100 * i / pow(2, 16));
        QByteArray guess = given_key + guess_part;
        QByteArray check = QCryptographicHash::hash(pass + guess, QCryptographicHash::Sha3_256);
        if(check == verification) {
            bytes_key = guess;
            break;
        }
    }
    if(bytes_key.isEmpty()) {
        fail("veriffail");
        return nullptr;
    }

    QByteArray pass_rand = QCryptographicHash::hash(pass + bytes_key, QCryptographicHash::Sha3_512);

    long long raw_size = mod(heavy_data_size[3]) +
            mod(heavy_data_size[2]) * pow(2, 8) +
            mod(heavy_data_size[1]) * pow(2, 16) +
            mod(heavy_data_size[0]) * pow(2, 24);
    emit setProgress(0);
    proccessPixelsv1_4(image, &heavy_data, pass_rand, false, were, raw_size);
    QByteArray unzipped = unzip(heavy_data, pass_rand);
    emit setProgress(101);
    QTime final = QTime::currentTime();
    qDebug() << "[Debug] Finished decrypting in " << st.msecsTo(final) << " msecs.";
    return unzipped;
}
void ModelPC::proccessPixelsv1_4(QImage *image, QByteArray*
data, QByteArray key, bool isEncrypt, QVector <QPair<QPoint, QPair<int, int>>> *were, long long size
)
{
    long w = image->width();
    long h = image->height();
    auto seed_hex = QCryptographicHash::hash(key, QCryptographicHash::Sha3_256).toHex().left(8).toUpper();
    auto seed = seed_hex.toLongLong(nullptr, 16);
    QRandomGenerator foo(seed);

    bitsBuffer.clear();
    long long left = (size == -1 ? data->size() : size) * 8;
    long long all = left;
    long cur = 0;
    if(isEncrypt) {
        while(left > 0 && success)
        {
            if(bitsBuffer.empty())
                push(mod(data->at(cur++)), 8);
            quint64 g = foo.generate64() % (w * h);
            long x = g % w;
            long y = g / w;
            int c = foo.generate64() % 3;
            int b = foo.generate64() % 24;
            int bit = -1;
            if(b < 16)
                bit = 7;
            else if(bit < 20)
                bit = 6;
            else if(bit < 22)
                bit = 5;
            else if(bit < 23)
                bit = 4;
            else if(bit < 24)
                bit = 3;
            auto piece = qMakePair(QPoint(x, y), qMakePair(c, bit));
            if(were->contains(piece))
                continue;
            were->append(piece);
            left--;
            emit setProgress(100 * (all - left) / all);
            int wr = pop(1);
            QColor pixel = image->pixelColor(piece.first);
            int red = pixel.red();
            int green = pixel.green();
            int blue = pixel.blue();
            int dif;
            if(c == 0)
                dif = red;
            else if (c == 1)
                dif = green;
            else
                dif = blue;
            dif |= 1 << (7 - bit);
            dif ^= (wr ^ 1) << (7 - bit);
            if(c == 0)
                red = dif;
            else if(c == 1)
                green = dif;
            else
                blue = dif;
            image->setPixelColor(piece.first, QColor(red, green, blue));
        }
    } else {
        while(left > 0)
        {
            while (bitsBuffer.size() >= 8)
                data->push_back(pop(8));
            quint64 g = foo.generate64() % (w * h);
            long x = g % w;
            long y = g / w;
            int c = foo.generate64() % 3;
            int b = foo.generate64() % 24;
            int bit = -1;
            if(b < 16)
                bit = 7;
            else if(bit < 20)
                bit = 6;
            else if(bit < 22)
                bit = 5;
            else if(bit < 23)
                bit = 4;
            else if(bit < 24)
                bit = 3;
            auto piece = qMakePair(QPoint(x, y), qMakePair(c, bit));
            if(were->contains(piece))
                continue;
            were->append(piece);
            left--;
            emit setProgress(100 * (all - left) / all);
            QColor pixel = image->pixelColor(piece.first);
            int red = pixel.red();
            int green = pixel.green();
            int blue = pixel.blue();
            int dif;
            if(c == 0)
                dif = red;
            else if (c == 1)
                dif = green;
            else
                dif = blue;
            dif &= 1 << (7 - bit);
            int wr = dif != 0;
            push(wr, 1);
        }
        while (bitsBuffer.size() >= 8)
            data->push_back(pop(8));
    }
}

QByteArray ModelPC::decryptv1_3(QImage *image, QString key)
{
    // Image opening
    int w = image->width();
    int h = image->height();

    // Getting corner pixels
    QColor colUL = image->pixelColor(0, 0).toRgb();
    QColor colUR = image->pixelColor(w - 1, 0).toRgb();
    QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();


    // Getting verification code
    int verifCode = (((colUR.green() % 2) << 5) + colUR.blue() % 32) << 2;
    verifCode += colDR.blue() % 4;
    if(verifCode != 166){
        fail("veriffail");
        return nullptr;
    }
    // Getting number of bytes
    long long int countBytes = (colUL.blue() % 32 + ((colUL.green() % 32) << 5) + ((colUL.red() % 32) << 10
)) << 9;
    countBytes += ((colUR.red() % 32) << 4) + (colUR.green() >> 1) % 16;

    bitsUsed = (colDR.blue() >> 2) % 8 + 1;
    // curMode = colDR.green() % 32;

    // Start of the circuit
    QByteArray data;
    circuit(image, &data, countBytes);

    // Check if circuit was successful
    if(!success)
        return nullptr;
    if(data.isEmpty())
    {
        fail("noreaddata");
        return nullptr;

    }
    // Version check
    long long int _ver = mod(data.at(0)) * qPow(2, 16);
    _ver += mod(data.at(1)) * qPow(2, 8);
    _ver += mod(data.at(2));
    data.remove(0, 3);
    if(_ver > version) {
        fail("new_version");
        return nullptr;
    }
    else if(_ver < version) {
        fail("old_version");
        return nullptr;
    }
    // Get the hash
    QByteArray hash = data.left(32);
    data.remove(0, 32);

    // Unzip
    QByteArray unzipped_data = unzip(data, key.toUtf8());
    QByteArray our_hash = QCryptographicHash::hash(unzipped_data, QCryptographicHash::Sha256);
    if(our_hash != hash) {
        fail("veriffail");
        return QByteArray("");
    }
    return unzipped_data;
}
long ModelPC::pop(int bits)
{
    // Hard to say
    long res = 0;
    int poppedBits = bits == -1 ? bitsUsed : bits;
    if(bits == -2)
        poppedBits = bitsBuffer.size();
    for(int i = 0; i < poppedBits; i++)
        res += bitsBuffer[i] * qPow(2, poppedBits - i - 1);
    bitsBuffer.remove(0, poppedBits);
    return res;
}

void ModelPC::push(int data, int bits)
{
    // That's easier, but also hard
    int buf_size = bitsBuffer.size();
    int extraSize = bits == -1 ? bitsUsed : bits;
    bitsBuffer.resize(buf_size + extraSize);
    for(int i = bitsBuffer.size() - 1; i >= buf_size; i--, data >>= 1)
        bitsBuffer[i] = data % 2;
}

bool ModelPC::mustGoOn(bool isEncrypt)
{
    return success && (isEncrypt ? (circuitCountBytes - cur) * 8 + bitsBuffer.size() >= bitsUsed * 3
 :
                                   circuitData->size() * 8 + bitsBuffer.size() <
                                   circuitCountBytes * 8 - (circuitCountBytes * 8)% (bitsUsed * 3));
}
QByteArray ModelPC::unzip(QByteArray data, QByteArray key)
{
    // Decryption
    QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
    QAESEncryption encryption(QAESEncryption::AES_256,
QAESEncryption::ECB);
    QByteArray new_data = encryption.decode(data, hashKey);
    // Decompressing
    return qUncompress(new_data);
}
QByteArray ModelPC::zip(QByteArray data, QByteArray key)
{
    // Zip
    QByteArray c_data = qCompress(data, 9);
    // Encryption
    QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
    return QAESEncryption::Crypt(QAESEncryption::AES_256,
QAESEncryption::ECB, c_data, hashKey);
}

bool ModelPC::fileExists(QString path)
{
    QFileInfo check_file(path);
    return check_file.exists() && check_file.isFile();
}

QByteArray ModelPC::bytes(long long n)
```

```
{
    return QByteArray::fromHex(QByteArray::number(n, 16));
}
unsigned int ModelPC::mod(int input)
{
    if(input < 0)
        return (unsigned int) (256 + input);
    else
        return (unsigned int) input;
}
void ModelPC::alert(QString message, bool isWarning)
{
    emit alertView(message, isWarning);
}
QColor ModelPC::RGBbytes(long long byte)
{
    int blue = byte % 256;
    int green = (byte / 256) % 256;
    int red = byte / qPow(2, 16);
    return QColor(red, green, blue);
}

QString ModelPC::generateVersionString(long ver)
{
    return QString::number((int)( ver / qPow(2, 16))) + "." + QString::number(((int) (ver / 256)) % 256) +
"." + QString::number(ver % 256);
}

uint ModelPC::randSeed()
{
    QTime time = QTime::currentTime();
    uint randSeed = time.msecsSinceStartOfDay() % 55363 + time.minute() * 21 + time.second() * 2 + 239;
    qsrand(randSeed);
    uint randSeed_2 = qrand() % 72341 + qrand() % 3 + qrand() % 2 + 566;
    return randSeed_2;
}
QByteArray ModelPC::GetRandomBytes(long long count)
{
    QByteArray res;
    for(int i = 0; i < count; i++)
       res.append(qrand() % 256);
    return res;
}
```

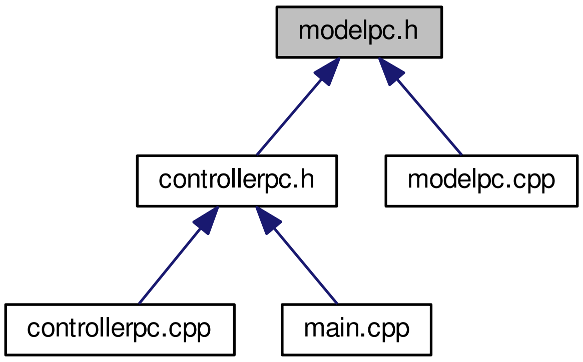
8.21 modelpc.h File Reference

```
#include <QObject>
#include <QImage>
#include <QByteArray>
#include <QColor>
#include <QPoint>
#include <QVector>
#include <QProcess>
#include <QTime>
#include <QFileInfo>
#include <QtGui>
#include <QtCore/QRandomGenerator>
#include <QPair>
#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

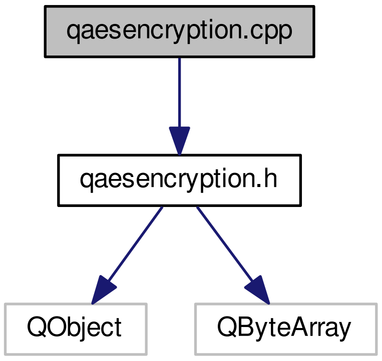
```
#ifndef MODELPC_H
#define MODELPC_H

#include <QObject>
#include <QImage>
#include <QByteArray>
#include <QColor>
#include <QPoint>
#include <QVector>
#include <QProcess>
#include <QTime>
#include <QFileInfo>
#include <QtGui>
#include <QtCore/QRandomGenerator>
#include <QPair>

#include "aes/qaesencryption.h"
#include <QCryptographicHash>
```

```


class ModelPC : public QObject
{
    Q_OBJECT
public:
    ModelPC();
    enum CryptMode {NotDefined, v1_3, v1_4, jphs_mode};
    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);

signals:
    void alertView(QString messageCode, bool isWarning);
    void saveData(QByteArray data);
    void saveImage(QImage *image);
    void setProgress(int val);

public slots:
    QImage *encrypt(QByteArray data, QImage *image, int _mode, QString key = "", int _bitsUsed = 8, QString *_error = nullptr);
    QImage *inject(QByteArray encr_data, QImage * image, int _mode, int _bitsUsed = 8, QString * _error = nullptr);
    QByteArray decrypt(QImage * image, QString key, int _mode = 0, QString *_error = nullptr);
    void fail(QString message);
    void alert(QString message, bool isWarning = false);

public:
    QByteArray unzip(QByteArray data, QByteArray key);

    bool success;
    long version;
    QString versionString;
    QString defaultJPHSDir;
protected:
    void circuit(QImage * image, QByteArray * data, long long int countBytes);
    void jphs(QImage * image, QByteArray * data);
    void processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt);
    void encryptv1_4(QImage *image, QByteArray data, QString key);
    QByteArray decryptv1_3(QImage * image, QString key);
    QByteArray decryptv1_4(QImage * image, QString key);
    void proccessPixelsv1_4(QImage *image, QByteArray* data, QByteArray key, bool isEncrypt, QVector<QPair<QPoint, QPair<int, int> > > *were, long long size = -1);
    QByteArray zip(QByteArray data, QByteArray key);

    QString * error;
private:
    int bitsUsed;
    bool fileExists(QString path);
    QByteArray bytes(long long n);
    unsigned int mod(int input);
    QByteArray ver_byte;
    QColor RGBbytes(long long byte);
    QString generateVersionString(long ver);
    uint randSeed();
    bool isTry = false;

    QByteArray * circuitData;
    QImage * circuitImage;
    long long circuitCountBytes;
    long cur;
    bool mustGoOn(bool isEncrypt);

    QVector <bool> bitsBuffer;
    long pop(int bits = -1);
    void push(int data, int bits = -1);

    void setError(QString word);
    QByteArray GetRandomBytes(long long count = 32);
};

#endif // MODELPC_H
```

8.23 qaesencryption.cpp File Reference

```
#include "qaesencryption.h"
```

Include dependency graph for qaesencryption.cpp:



Functions

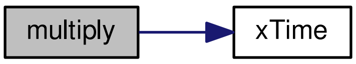
- quint8 xTime (quint8 x)
- quint8 multiply (quint8 x, quint8 y)

8.23.1 Function Documentation

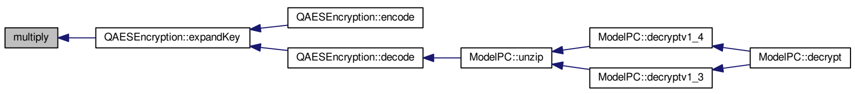
8.23.1.1 quint8 multiply (quint8 *x*, quint8 *y*) `[inline]`

Definition at line 57 of file qaesencryption.cpp.

Here is the call graph for this function:



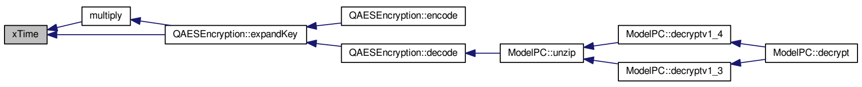
Here is the caller graph for this function:



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

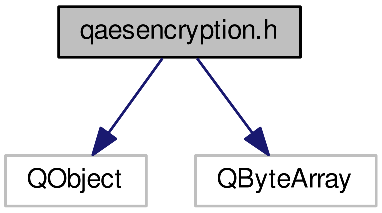
```
00001 #include "qaesencryption.h"
00002
00003 /*
00004  * Static Functions
00005  * */
00006 QByteArray QAESEncryption::Crypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText,
00007                                  const QByteArray &key, const QByteArray &iv,
      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));
```

```
}

inline quint8 multiply(quint8 x, quint8 y){
  return (((y & 1) * x) ^ ((y>>1 & 1) * xTime(x)) ^ ((y>>2 & 1) * xTime(
xTime(x))) ^ ((y>>3 & 1)
            * xTime(xTime(xTime(x)))) ^ ((y>>4 & 1) * xTime(
xTime(xTime(xTime(x))))));
}

/*
 * End Inline functions
 * */


QAESEncryption::QAESEncryption(Aes level, Mode mode,
                               Padding padding)
    : m_nb(4), m_blocklen(16), m_level(level), m_mode(mode), m_padding(padding)
{
    m_state = NULL;

    switch (level)
    {
    case AES_128: {
        AES128 aes;
        m_nk = aes.nk;
        m_keyLen = aes.keylen;
        m_nr = aes.nr;
        m_expandedKey = aes.expandedKey;
        }
        break;
    case AES_192: {
        AES192 aes;
        m_nk = aes.nk;
        m_keyLen = aes.keylen;
        m_nr = aes.nr;
        m_expandedKey = aes.expandedKey;
        }
        break;
    case AES_256: {
        AES256 aes;
        m_nk = aes.nk;
        m_keyLen = aes.keylen;
        m_nr = aes.nr;
        m_expandedKey = aes.expandedKey;
        }
        break;
    default: {
        AES128 aes;
        m_nk = aes.nk;
        m_keyLen = aes.keylen;
        m_nr = aes.nr;
        m_expandedKey = aes.expandedKey;
        }
        break;
    }

}
QByteArray QAESEncryption::getPadding(int currSize, int alignment)
{
    int size = (alignment - currSize % alignment) % alignment;
    if (size == 0) return QByteArray();
    switch(m_padding)
    {
    case Padding::ZERO:
        return QByteArray(size, 0x00);
        break;
    case Padding::PKCS7:
        return QByteArray(size,size);
        break;
    case Padding::ISO:
        return QByteArray (size-1, 0x00).prepend(0x80);
        break;
    default:
        return QByteArray(size, 0x00);
        break;
    }
    return QByteArray(size, 0x00);
}

QByteArray QAESEncryption::expandKey(const QByteArray &
key)
{
  int i, k;
  quint8 tempa[4]; // Used for the column/row operations
  QByteArray roundKey(key);

  // The first round key is the key itself.
  // ...

  // All other round keys are found from the previous round keys.
  //i == Nk
  for(i = m_nk; i < m_nb * (m_nr + 1); i++)
  {
    tempa[0] = (quint8) roundKey.at((i-1) * 4 + 0);
    tempa[1] = (quint8) roundKey.at((i-1) * 4 + 1);
    tempa[2] = (quint8) roundKey.at((i-1) * 4 + 2);
    tempa[3] = (quint8) roundKey.at((i-1) * 4 + 3);

    if (i % m_nk == 0)
    {
        // This function shifts the 4 bytes in a word to the left once.
        // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

        // Function RotWord()
        k = tempa[0];
        tempa[0] = tempa[1];
        tempa[1] = tempa[2];
        tempa[2] = tempa[3];
        tempa[3] = k;

        // Function Subword()
        tempa[0] = getSBoxValue(tempa[0]);
        tempa[1] = getSBoxValue(tempa[1]);
        tempa[2] = getSBoxValue(tempa[2]);
        tempa[3] = getSBoxValue(tempa[3]);

        tempa[0] =  tempa[0] ^ Rcon[i/m_nk];
    }
    if (m_level == AES_256 && i % m_nk == 4)
    {
        // Function Subword()
        tempa[0] = getSBoxValue(tempa[0]);
        tempa[1] = getSBoxValue(tempa[1]);
        tempa[2] = getSBoxValue(tempa[2]);
        tempa[3] = getSBoxValue(tempa[3]);
    }
    roundKey.insert(i * 4 + 0, (quint8) roundKey.at((i - m_nk) * 4 + 0) ^ tempa[0]);
    roundKey.insert(i * 4 + 1, (quint8) roundKey.at((i - m_nk) * 4 + 1) ^ tempa[1]);
    roundKey.insert(i * 4 + 2, (quint8) roundKey.at((i - m_nk) * 4 + 2) ^ tempa[2]);
    roundKey.insert(i * 4 + 3, (quint8) roundKey.at((i - m_nk) * 4 + 3) ^ tempa[3]);
  }
  return roundKey;
}

// This function adds the round key to state.
// The round key is added to the state by an XOR function.
void QAESEncryption::addRoundKey(const quint8 round, const QByteArray expKey)
{
  QByteArray::iterator it = m_state->begin();
  for(int i=0; i < 16; ++i)
      it[i] = (quint8) it[i] ^ (quint8) expKey.at(round * m_nb * 4 + (i/4) * m_nb + (i%4));
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void QAESEncryption::subBytes()
{
  QByteArray::iterator it = m_state->begin();
  for(int i = 0; i < 16; i++)
    it[i] = getSBoxValue((quint8) it[i]);
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
void QAESEncryption::shiftRows()
{
    QByteArray::iterator it = m_state->begin();
    quint8 temp;
    //Keep in mind that QByteArray is column-driven!!

     //Shift 1 to left
    temp   = (quint8)it[1];
    it[1]  = (quint8)it[5];
    it[5]  = (quint8)it[9];
    it[9]  = (quint8)it[13];
    it[13] = (quint8)temp;

    //Shift 2 to left
    temp   = (quint8)it[2];
    it[2]  = (quint8)it[10];
    it[10] = (quint8)temp;
    temp   = (quint8)it[6];
    it[6]  = (quint8)it[14];
    it[14] = (quint8)temp;

    //Shift 3 to left
    temp   = (quint8)it[3];
    it[3]  = (quint8)it[15];
    it[15] = (quint8)it[11];
    it[11] = (quint8)it[7];
    it[7]  = (quint8)temp;
}

// MixColumns function mixes the columns of the state matrix
//optimized!!
void QAESEncryption::mixColumns()
{
  QByteArray::iterator it = m_state->begin();
  quint8 tmp, tm, t;

  for(int i = 0; i < 16; i += 4){
    t       = (quint8)it[i];
    tmp     =  (quint8)it[i] ^ (quint8)it[i+1] ^ (quint8)it[i+2] ^ (quint8)it[i+3] ;

    tm      = xTime( (quint8)it[i] ^ (quint8)it[i+1] );
    it[i]   = (quint8)it[i] ^ (quint8)tm ^ (quint8)tmp;

    tm      = xTime( (quint8)it[i+1] ^ (quint8)it[i+2]);
    it[i+1] = (quint8)it[i+1] ^ (quint8)tm ^ (quint8)tmp;

    tm      = xTime( (quint8)it[i+2] ^ (quint8)it[i+3]);
    it[i+2] =(quint8)it[i+2] ^ (quint8)tm ^ (quint8)tmp;

    tm      = xTime((quint8)it[i+3] ^ (quint8)t);
    it[i+3] =(quint8)it[i+3] ^ (quint8)tm ^ (quint8)tmp;
  }
}

// MixColumns function mixes the columns of the state matrix.
// The method used to multiply may be difficult to understand for the inexperienced.
// Please use the references to gain more information.
void QAESEncryption::invMixColumns()
{
  QByteArray::iterator it = m_state->begin();
  quint8 a,b,c,d;
  for(int i = 0; i < 16; i+=4){
    a = (quint8) it[i];
    b = (quint8) it[i+1];
    c = (quint8) it[i+2];
    d = (quint8) it[i+3];

    it[i]   = (quint8) (multiply(a, 0x0e) ^ multiply(b, 0x0b) ^
multiply(c, 0x0d) ^ multiply(d, 0x09));
    it[i+1] = (quint8) (multiply(a, 0x09) ^ multiply(b, 0x0e) ^
multiply(c, 0x0b) ^ multiply(d, 0x0d));
    it[i+2] = (quint8) (multiply(a, 0x0d) ^ multiply(b, 0x09) ^
multiply(c, 0x0e) ^ multiply(d, 0x0b));
    it[i+3] = (quint8) (multiply(a, 0x0b) ^ multiply(b, 0x0d) ^
multiply(c, 0x09) ^ multiply(d, 0x0e));
  }
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void QAESEncryption::invSubBytes()
{
    QByteArray::iterator it = m_state->begin();
    for(int i = 0; i < 16; ++i)
        it[i] = getSBoxInvert((quint8) it[i]);
}

void QAESEncryption::invShiftRows()
{
    QByteArray::iterator it = m_state->begin();
    uint8_t temp;

    //Keep in mind that QByteArray is column-driven!!

    //Shift 1 to right
    temp   = (quint8)it[13];
    it[13] = (quint8)it[9];
    it[9]  = (quint8)it[5];
    it[5]  = (quint8)it[1];
    it[1]  = (quint8)temp;

    //Shift 2
    temp   = (quint8)it[10];
    it[10] = (quint8)it[2];
    it[2]  = (quint8)temp;
    temp   = (quint8)it[14];
    it[14] = (quint8)it[6];
    it[6]  = (quint8)temp;

    //Shift 3
    temp   = (quint8)it[15];
    it[15] = (quint8)it[3];
    it[3]  = (quint8)it[7];
    it[7]  = (quint8)it[11];
    it[11] = (quint8)temp;
}

QByteArray QAESEncryption::byteXor(const QByteArray &a, const QByteArray &b)
{
  QByteArray::const_iterator it_a = a.begin();
  QByteArray::const_iterator it_b = b.begin();
  QByteArray ret;

  //for(int i = 0; i < m_blocklen; i++)
  for(int i = 0; i < std::min(a.size(), b.size()); i++)
      ret.insert(i,it_a[i] ^ it_b[i]);

  return ret;
}

// Cipher is the main function that encrypts the PlainText.
QByteArray QAESEncryption::cipher(const QByteArray &expKey, const QByteArray &in)
{

  //m_state is the input buffer...
  QByteArray output(in);
  m_state = &output;

  // Add the First round key to the state before starting the rounds.
  addRoundKey(0, expKey);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr-1 rounds are executed in the loop below.
  for(quint8 round = 1; round < m_nr; ++round){
    subBytes();
    shiftRows();
    mixColumns();
    addRoundKey(round, expKey);
  }

  // The last round is given below.
  // The MixColumns function is not here in the last round.
  subBytes();
  shiftRows();
  addRoundKey(m_nr, expKey);

  return output;
}

QByteArray QAESEncryption::invCipher(const QByteArray &expKey, const QByteArray &in)
{
    //m_state is the input buffer.... handle it!
    QByteArray output(in);
    m_state = &output;

    // Add the First round key to the state before starting the rounds.
    addRoundKey(m_nr, expKey);

    // There will be Nr rounds.
    // The first Nr-1 rounds are identical.
    // These Nr-1 rounds are executed in the loop below.
    for(quint8 round=m_nr-1; round>0 ; round--){
        invShiftRows();
        invSubBytes();
        addRoundKey(round, expKey);
        invMixColumns();
    }

    // The last round is given below.
    // The MixColumns function is not here in the last round.
    invShiftRows();
    invSubBytes();
    addRoundKey(0, expKey);

    return output;
}

QByteArray QAESEncryption::encode(const QByteArray &rawText, const QByteArray &
key, const QByteArray &iv)
{
    if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
       return QByteArray();
    QByteArray ret;
    QByteArray expandedKey = expandKey(key);
    QByteArray alignedText(rawText);

    //Fill array with padding
    alignedText.append(getPadding(rawText.size(), m_blocklen));

    switch(m_mode)
    {
    case ECB:
        for(int i=0; i < alignedText.size(); i+= m_blocklen)
            ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
        break;
    case CBC: {
            QByteArray ivTemp(iv);
            for(int i=0; i < alignedText.size(); i+= m_blocklen) {
                alignedText.replace(i, m_blocklen, byteXor(alignedText.mid(i, m_blocklen),ivTemp));
                ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
                ivTemp = ret.mid(i, m_blocklen);
            }
        }
        break;
    case CFB: {
            ret.append(byteXor(alignedText.left(m_blocklen), cipher(expandedKey, iv)));
            for(int i=0; i < alignedText.size(); i+= m_blocklen) {
                if (i+m_blocklen < alignedText.size())
                    ret.append(byteXor(alignedText.mid(i+m_blocklen, m_blocklen),
                                       cipher(expandedKey, ret.mid(i, m_blocklen))));
            }
        }
        break;
    case OFB: {
            QByteArray ofbTemp;
            ofbTemp.append(cipher(expandedKey, iv));
            for (int i=m_blocklen; i < alignedText.size(); i += m_blocklen){
                ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
            }
            ret.append(byteXor(alignedText, ofbTemp));
        }
        break;
    default: break;
    }
    return ret;
}

QByteArray QAESEncryption::decode(const QByteArray &rawText, const QByteArray &
key, const QByteArray &iv)
{
    if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
       return QByteArray();

    QByteArray ret;
    QByteArray expandedKey = expandKey(key);

    switch(m_mode)
    {
    case ECB:
        for(int i=0; i < rawText.size(); i+= m_blocklen)
            ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
        break;
    case CBC: {
            QByteArray ivTemp(iv);
            for(int i=0; i < rawText.size(); i+= m_blocklen){
                ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
                ret.replace(i, m_blocklen, byteXor(ret.mid(i, m_blocklen),ivTemp));
                ivTemp = rawText.mid(i, m_blocklen);
            }
        }
        break;
    case CFB: {
            ret.append(byteXor(rawText.mid(0, m_blocklen), cipher(expandedKey, iv)));
            for(int i=0; i < rawText.size(); i+= m_blocklen){
                if (i+m_blocklen < rawText.size()) {
                    ret.append(byteXor(rawText.mid(i+m_blocklen, m_blocklen),
                                       cipher(expandedKey, rawText.mid(i, m_blocklen))));
                }
            }
        }
        break;
    case OFB: {
        QByteArray ofbTemp;
        ofbTemp.append(cipher(expandedKey, iv));
        for (int i=m_blocklen; i < rawText.size(); i += m_blocklen){
            ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
        }
        ret.append(byteXor(rawText, ofbTemp));
    }
        break;
    default:
        //do nothing
        break;
    }
    return ret;
}

QByteArray QAESEncryption::removePadding(const QByteArray &rawText)
{
    QByteArray ret(rawText);
    switch (m_padding)
    {
    case Padding::ZERO:
        //Works only if the last byte of the decoded array is not zero
        while (ret.at(ret.length()-1) == 0x00)
            ret.remove(ret.length()-1, 1);
        break;
    case Padding::PKCS7:
        ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
        break;
    case Padding::ISO:
        ret.truncate(ret.lastIndexOf(0x80));
        break;
    default:
        //do nothing
        break;
    }
    return ret;
}
```

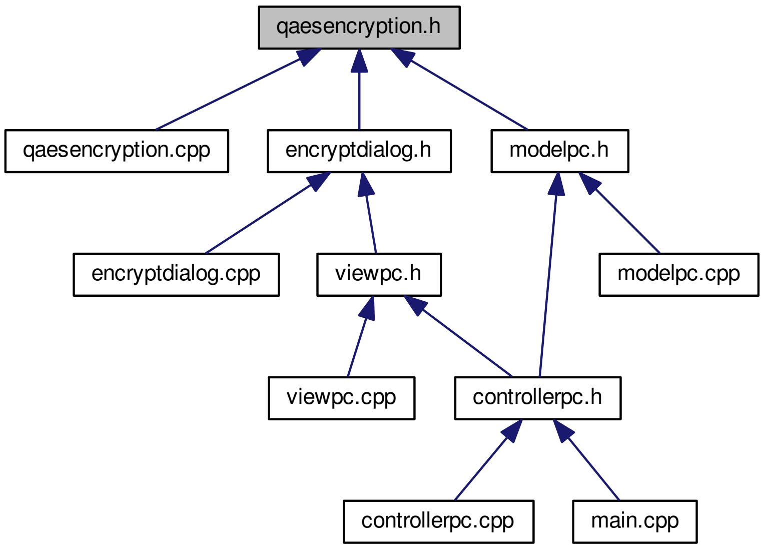
8.25 qaesencryption.h File Reference

```
#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`.

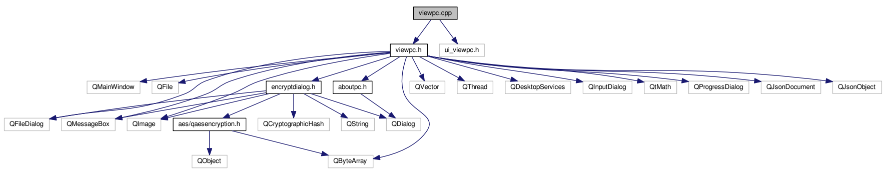
8.26 qaesencryption.h

```
#ifndef QAESENCRYPTION_H
#define QAESENCRYPTION_H

#include <QObject>
#include <QByteArray>

class QAESEncryption : public QObject
{
    Q_OBJECT
public:
    enum Aes {
        AES_128,
        AES_192,
        AES_256
    };
    enum Mode {
        ECB,
        CBC,
        CFB,
        OFB
    };

    enum Padding {
      ZERO,
      PKCS7,
      ISO
    };
    static QByteArray Crypt(QAESEncryption::Aes level,
QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
                            const QByteArray &iv = NULL, QAESEncryption::Padding padding = QAESEncryption::ISO);
    static QByteArray Decrypt(QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
                              const QByteArray &iv = NULL, QAESEncryption::Padding padding = QAESEncryption::ISO);
    static QByteArray ExpandKey(QAESEncryption::Aes level, QAESEncryption::Mode mode, const QByteArray &key);
    static QByteArray RemovePadding(const QByteArray &rawText, QAESEncryption::Padding padding);

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

signals:

public slots:

private:
    int m_nb;
    int m_blocklen;
    int m_level;
    int m_mode;
    int m_nk;
    int m_keyLen;
    int m_nr;
    int m_expandedKey;
    int m_padding;
    QByteArray* m_state;

    struct AES256{
        int nk = 8;
        int keylen = 32;
        int nr = 14;
        int expandedKey = 240;
    };

    struct AES192{
        int nk = 6;
        int keylen = 24;
        int nr = 12;
        int expandedKey = 209;
    };

    struct AES128{
        int nk = 4;
        int keylen = 16;
        int nr = 10;
        int expandedKey = 176;
    };

    quint8 getSBoxValue(quint8 num){return sbox[num];}
    quint8 getSBoxInvert(quint8 num){return rsbox[num];}

    void addRoundKey(const quint8 round, const QByteArray expKey);
    void subBytes();
    void shiftRows();
    void mixColumns();
    void invMixColumns();
    void invSubBytes();
    void invShiftRows();
    QByteArray getPadding(int currSize, int alignment);
    QByteArray cipher(const QByteArray &expKey, const QByteArray &plainText);
    QByteArray invCipher(const QByteArray &expKey, const QByteArray &plainText);
    QByteArray byteXor(const QByteArray &in, const QByteArray &iv);

    const quint8 sbox[256] =   {
      //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
      0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
      0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
      0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
      0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
      0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
      0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
      0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
      0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
      0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
      0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
      0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
      0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
      0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
      0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
      0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
      0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };

    const quint8 rsbox[256] =
    { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
      0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
      0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
      0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
      0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
      0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
      0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
      0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
      0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
      0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
      0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
      0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
      0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
      0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
      0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
      0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };

    // The round constant word array, Rcon[i], contains the values given by
    // x to th e power (i-1) being powers of x (x is denoted as {02}) in the field GF(2^8)
    // Only the first 14 elements are needed
    const quint8 Rcon[256] = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab/*, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
        0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
        0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
        0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
        0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
        0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
        0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
            0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
*/};
};

#endif // QAESENCRYPTION_H
```

8.27 viewpc.cpp File Reference

```
#include "viewpc.h"
#include "ui_viewpc.h"
```

Include dependency graph for viewpc.cpp:



8.28 viewpc.cpp

```
#include "viewpc.h"
```

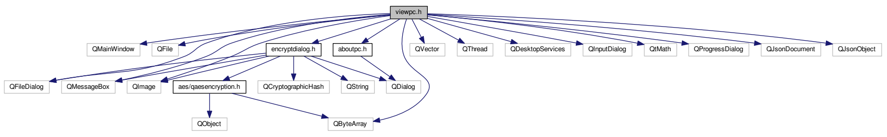
```
#include "ui_viewpc.h"

ViewPC::ViewPC(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::ViewPC)
{
    ui->setupUi(this);

    progressDialogClosed = true;

    // Alerts dictionary setup
    QFile file(":/config/ErrorsDict.json");
    if(!file.open(QFile::ReadOnly | QFile::Text)) {
        alert("Cannot open config file!");
        return;
    }
    QByteArray readData = file.readAll();
    file.close();

    QJsonParseError error;
    QJsonDocument doc = QJsonDocument::fromJson(readData, &error);
    errorsDict = doc.object();
    isEncrypt = true;
}
ViewPC::~ViewPC()
{
    delete ui;
}

void ViewPC::on_encryptMode_clicked()
{
    // Encrypt radio button clicked
    setEncryptMode(true);
}

void ViewPC::on_decryptMode_clicked()
{
    // Decrypt radio button clicked
    setEncryptMode(false);
}
void ViewPC::on_fileButton_clicked()
{
    // Opening QFileDialog depending on isEncrypt
    if(isEncrypt)
        inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.txt", tr("Text
    files (*.txt);;All Files (*)"));
    else
        inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.png", tr("PNG
    files (*.png);;All Files (*)"));
    // Display the file name
    ui->fileLabel->setText(inputFileName.isEmpty() ? "File not chosen" : inputFileName);
}
void ViewPC::on_startButton_clicked()
{
    if(isEncrypt)
    {
        // Getting the data
        QString text = ui->text->toPlainText();
        QByteArray data;
        if(text.isEmpty()) {
            if(inputFileName.isEmpty()) {
                alert("no_input_file", true);
                return;
            }
            // Opening the file
            QFile file(inputFileName);
            if (!file.open(QIODevice::ReadOnly))
            {
                alert("open_file_fail", true);
                return;
            }
            // Check the data size
            auto size = file.size();
            if(size > qPow(2, 24)) {
                alert("muchdata", true);
                file.close();
                return;
            }
            data = file.readAll();
            file.close();
        }
        else
            data = text.toUtf8();
        // Select image via EncryptDialog
        EncryptDialog * dialog = new EncryptDialog(data);
        dialog->exec();
        if(!dialog->success)
```

```
            return;

        // Get the data
        QByteArray encr_data = dialog->compr_data;

        // Save the hash
        QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);
        encr_data = hash + encr_data;

        switch (selectedMode) {
        case 1:
            emit inject(encr_data, &dialog->image, selectedMode, dialog->
bitsUsed);
            break;
        case 2:
            emit encrypt(data, &dialog->image, selectedMode, dialog->
key);
            break;
        }
    }
    else
    {
        // Get the filename of the image
        if(inputFileName.isEmpty()) {
            alert("no_input_file", true);
            return;
        }
        QByteArray key = requestKey().toUtf8();
        if(key.isEmpty())
            return;
        QImage * res_image = new QImage(inputFileName);
        emit decrypt(res_image, key, 0);
    }
}
void ViewPC::alert(QString message, bool isWarning)
{
    // Get message
    if(errorsDict.contains(message))
        message = errorsDict[message].toString();
    // Create message box
    QMessageBox box;
    if(isWarning)
        box.setIcon(QMessageBox::Warning);
    else
        box.setIcon(QMessageBox::Information);
    box.setText(message);
    box.setWindowIcon(QIcon(":/icons/mail.png"));
    box.setWindowTitle("Message");
    box.exec();
}
void ViewPC::saveData(QByteArray Edata)
{
    // Save data using QFileDialog
    QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save File"),
                               "/untitled.txt",
                               tr("Text(*.txt);;All files (*)"));
    QFile writeFile(outputFileName);
    if (!writeFile.open(QIODevice::WriteOnly))
    {
        alert("save_file_fail", true);
        return;
    }
    writeFile.write(Edata);
    writeFile.close();
    alert("decryption_completed");
}
void ViewPC::saveImage(QImage * image)
{
    // Save image using QFileDialog
    QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save Image"),
                               "/untitled.png",
                               tr("Images(*.png)"));
    if(!image->save(outputFileName)) {
        alert("save_file_fail", true);
        return;
    }
    alert("encryption_completed");
}
void ViewPC::setProgress(int val)
{
    if(val < 0) {
        // Create dialog
        dialog = new QProgressDialog("Cryption in progress.", "Cancel", 0, 100);
        connect(dialog, SIGNAL(canceled()), this, SLOT(abortCircuit()));
        progressDialogClosed = false;
        dialog->setWindowTitle("Processing");
        dialog->setWindowIcon(QIcon(":/icons/loading.png"));
        dialog->show();
    }
    else if(val > 100 && !progressDialogClosed) {
        // Close dialog
        dialog->setValue(100);
        QThread::msleep(25);
        dialog->close();
        dialog->reset();
        progressDialogClosed = true;
    }
    // Update the progress
    else if(!progressDialogClosed)
        dialog->setValue(val);
}
void ViewPC::abortCircuit()
{
    // Set the flag
    progressDialogClosed = true;
    // Close the dialog
    dialog->close();
    dialog->reset();
    emit abortModel();
}
void ViewPC::setEncryptMode(bool encr)
{
    ui->text->setText("");
    ui->text->setEnabled(encr);
    isEncrypt = encr;
    ui->startButton->setText(encr ? "Continue configuration" : "Start decryption");
    ui->enLabel1->setText(encr ? "Type in the text for encryption:" : "Text input isn't supported in
 decryption mode");
    ui->enLabel1->setEnabled(encr);
    ui->enLabel2->setText(encr ? "Or use the file dialog to choose a file:" : "Choose a file for
 decryption:");
    ui->comboBox->setEnabled(encr);
}
void ViewPC::setVersion(QString version)
{
    // Version setup
    versionString = version;
}
QString ViewPC::requestKey()
{
    bool ok;
    QString text = QInputDialog::getText(this, tr("QInputDialog::getText()"),
                                         tr("Enter the keyphrase:"), QLineEdit::Normal,
                                         QDir::home().dirName(), &ok);
    if(text.isEmpty() && ok) {
        alert("no_key", true);
        return QString();
    }
    return ok ? text : QString();
}

QByteArray ViewPC::bytes(long long n)
{
    return QByteArray::fromHex(QByteArray::number(n, 16));
}
void ViewPC::on_actionAbout_triggered()
{
    AboutPC about;
    about.setVersion(versionString);
    about.exec();
}

void ViewPC::on_actionHelp_triggered()
{
    QUrl docLink("https://alexkovrigin.me/PictureCrypt");
    QDesktopServices::openUrl(docLink);
}

void ViewPC::on_actionJPHS_path_triggered()
{
    QString dir = QFileDialog::getExistingDirectory(this, tr("Open JPHS folder"),
                                                    "/home",
                                                    QFileDialog::ShowDirsOnly
                                                    | QFileDialog::DontResolveSymlinks);
    emit setJPHSDir(dir);
}

void ViewPC::on_comboBox_currentIndexChanged(int index)
{
    selectedMode = index + 1;
}

void ViewPC::on_text_textChanged()
{
    ui->fileButton->setEnabled(ui->text->toPlainText().isEmpty());
}
```

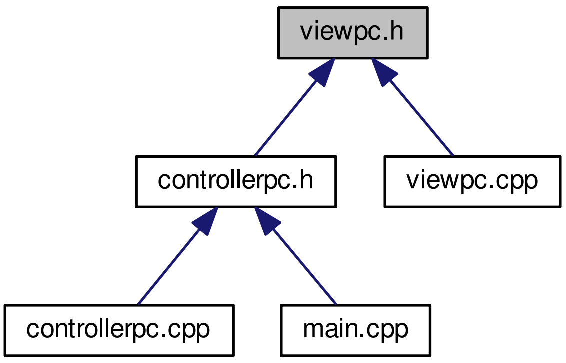
8.29 viewpc.h File Reference

```
#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

8.29.1 Detailed Description

Header of ViewPC class

See also

ControllerPC, ModelPC, ViewPC

Definition in file viewpc.h.

8.30 viewpc.h

```
#ifndef VIEWPC_H
#define VIEWPC_H

#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>

namespace Ui {
class ViewPC;
}
class ViewPC : public QMainWindow
{
    Q_OBJECT

public:
    explicit ViewPC(QWidget *parent = nullptr);
    ~ViewPC();
private slots:
    void on_encryptMode_clicked();

    void on_decryptMode_clicked();

    void on_actionJPHS_path_triggered();

    void on_comboBox_currentIndexChanged(int index);

    void on_text_textChanged();

protected slots:
    void on_fileButton_clicked();

    void on_startButton_clicked();

    void on_actionAbout_triggered();

    void on_actionHelp_triggered();
public slots:
    void alert(QString message, bool isWarning = false);
    void saveData(QByteArray Edata);
    void saveImage(QImage *image);
    void setProgress(int val);
    void abortCircuit();
    void setEncryptMode(bool encr);
    void setVersion(QString version);
signals:
    void encrypt(QByteArray data, QImage *image, int mode, QString key);
    void inject(QByteArray data, QImage * image, int mode, int bitsUsed);
    void decrypt(QImage * _image, QString key, int mode);
    void abortModel();
    void setJPHSDir(QString dir);
public:
    QProgressDialog * dialog;
    bool progressDialogClosed;
    QJsonObject errorsDict;
protected:
    QString requestKey();
private:
    Ui::ViewPC *ui;
    bool isEncrypt;
    QString inputFileName;
    QByteArray bytes(long long n);
    QString versionString;
    int selectedMode = 2;
};

#endif // VIEWPC_H
```

Index