PictureCrypt

1.2.5

Generated by Doxygen 1.8.14

Contents

Chapter 1

PictureCrypt

Project made using QT Creator on C++

1.1 The idea of the project

The idea came to me, when I read an article about steganoraphy. I realised, that you can store data in an image in pixels near the border, so noone can see and even if they did, it is practically impossible to decipher the contents.

1.2 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

1.3 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...

1.4 Structure of the project.

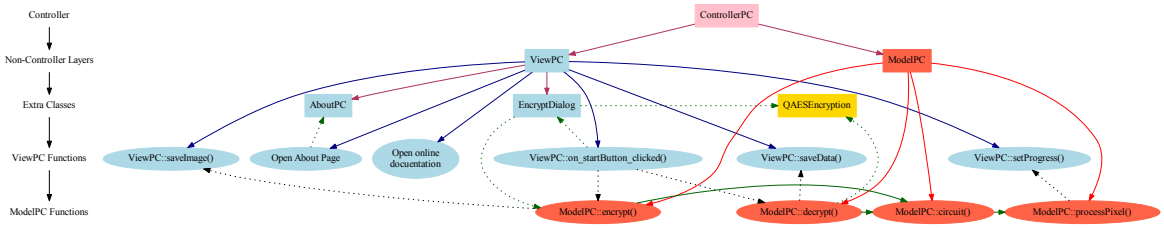
Project is done via MVC Pattern. View and Model layers are totally isolated and run on different threads.

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

1.5 External use

ModelPC class can be used externally (without UI)

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

...

ModelPC * model = new ModelPC(ver);
// ver is version of the app, used to check the data structure version
// ver is type long and is calculated as if version is "x.y.z" => ver = x * 65536 + y * 256 + z
// Default parameter is 2^17 (2.0.0)

// Connecting signals

// Essential ones

model->saveData(QByteArray data)
// Used to return the retrieved data

model->saveImage(QImage * image)
// Used to return the modified image

// Extra ones

model->alertView(QString message, bool isWarning)
// Used for messages to be shown to users

model->setProgress(int val)
// Used to show user the progress of embedding
// -1 indicates the creation of some kind of progress dialog
// from 0 to 100 shows the progress
// 101 indicates that progress dialog should be closed
```

See also

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

Avaible methods see here: https://waleko.github.io/PictureCrypt/#external-use or here ModelPC

1.6 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...

1.7 License

This software is provided under the UNLICENSE

1.8 Contact us

Visit our site: http://alex.unaux.com and Github Page: https://waleko.github.io Email me at a.kovrigin0@gmail.com

Author

Alex Kovrigin

Copyright

Chapter 2

PictureCrypt

Make your pictures crypted.

External usage

ModelPC class can be used externally (without UI)

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

...

ModelPC * model = new ModelPC(ver);
// ver is version of the app, used to check the data structure version
// ver is type long and is calculated as if version is "x.y.z" => ver = x * 65536 + y * 256 + z
// Default parameter is 2^17 (2.0.0)

// Connecting signals

// Essential ones

model->saveData(QByteArray data)
// Used to return the retrieved data

model->saveImage(QImage * image)
// Used to return the modified image

// Extra ones

model->alertView(QString message, bool isWarning)
// Used for messages to be shown to users

model->setProgress(int val)
// Used to show user the progress of embedding
// -1 indicates the creation of some kind of progress dialog
// from 0 to 100 shows the progress
// 101 indicates that progress dialog should be closed
```

Avaible methods

Essential ones

start

Used for embedding

Parameters: data Data to be encrypted _image Image to be encrypted into. _bitsUsed Bits per byte, see also ModelPC::bitsUsed key Key, if default (empty), random key of 64 charachters will be generated. mode Mode of encryption

```
model->start(QByteArray data, QImage image, int mode = 0, QString key = "", int _bitsUsed = 8);
```

decrypt

Used for de-embedding

Parameters: image Image to be decrypted.

```
model->decrypt(QImage * image);
```

Extra ones

encrypt

Used for embedding but with data already packed with stuff like version, file size, aes key, etc. Used in PictureCrypt project

Parameters:

encr_data Data to be embbed to an image. image Image to be embbed into. mode Mode of encryption

```
model->encrypt(QByteArray encr_data, QImage * image, int mode = 0);
```

fail

Used for stopping the embedding or de-embedding proccess Parameters:

message Message for user

```
model->fail(QString message);
```

Avaible modes of embedding

- 0 - Standard, created by me
- 1 - JPHS, requires manually installed JPHS and specified directory.

Documentation

Doxygen Documentation avaibale here

Windows Installer

Windows installer for non-QT build here

Dependancies

- qtcore
- QAESEncryption by bricke

Contact

Question or suggestions are welcome! Please use the GitHub issue tracking to report suggestions or issues. Email me a.kovrigin0@gmail.com and visit my site http://alex.unaux.com

License

This software is provided under the UNLICENSE

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 Ui Namespace Reference

Chapter 8

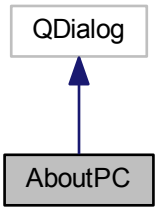
Class Documentation

8.1 AboutPC Class Reference

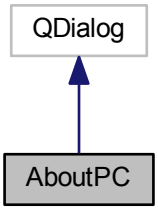
The AboutPC class The About Page dialog.

`#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.

8.1.1 Detailed Description

The AboutPC class The About Page dialog.

Definition at line 12 of file aboutpc.h.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 AboutPC()

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

Definition at line 4 of file aboutpc.cpp.

8.1.2.2 ~AboutPC()

```
AboutPC::~AboutPC ( )
```

Definition at line 11 of file aboutpc.cpp.

8.1.3 Member Function Documentation

8.1.3.1 setVersion()

```
void AboutPC::setVersion (
            QString version )
```

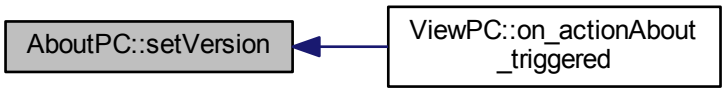
AboutPC::setVersion Function to set the version display.

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:

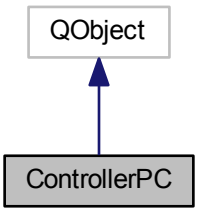
- C:/Users/salex/Documents/GitHub/PictureCrypt/aboutpc.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/aboutpc.cpp

8.2 ControllerPC Class Reference

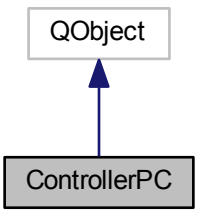
The ControllerPC class Controller class, which controls View and Model layers.

`#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 setBitsUsed (int bitsUsed)

  ControllerPC::setBitsUsed Slot to set ModelPC::bitsUsed.
- void setJPHSDir (QString dir)

  ControllerPC::setJPHSDir Sets JPHS default dir.

Public Member Functions

- ControllerPC ()

  ControllerPC::ControllerPC Constructor of controller Constructor checks the version of the app and 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.

8.2.1 Detailed Description

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

See also

ViewPC, ModelPC

Definition at line 19 of file controllerpc.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 ControllerPC()

```
ControllerPC::ControllerPC ( )
```

ControllerPC::ControllerPC Constructor of controller Constructor checks the version of the app and creates Model Class (ModelPC) and View Class (ViewPC). All signals and slots are connected here.

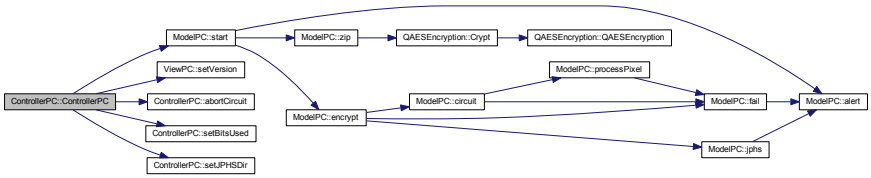
Controller class

Note

Version of the app is specified here.

Definition at line 9 of file controllerpc.cpp.

Here is the call graph for this function:



8.2.3 Member Function Documentation

8.2.3.1 abortCircuit

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

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

Definition at line 41 of file controllerpc.cpp.

Here is the caller graph for this function:

8.2.3.2 setBitsUsed

```
void ControllerPC::setBitsUsed (
            int bitsUsed )  [slot]
```

ControllerPC::setBitsUsed Slot to set ModelPC::bitsUsed.

Parameters

bitsUsed	Value

Definition at line 49 of file controllerpc.cpp.

Here is the caller graph for this function:

8.2.3.3 setJPHSDir

```
void ControllerPC::setJPHSDir (
            QString dir )  [slot]
```

ControllerPC::setJPHSDir Sets JPHS default dir.

Parameters

dir	Directory

Definition at line 57 of file controllerpc.cpp.

Here is the caller graph for this function:

8.2.4 Member Data Documentation

8.2.4.1 version

```
long int ControllerPC::version
```

version Version of the app

Definition at line 27 of file controllerpc.h.

8.2.4.2 versionString

```
QString ControllerPC::versionString
```

versionString Version of the app as QString.

Definition at line 31 of file controllerpc.h.

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

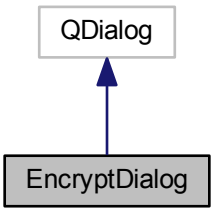
- C:/Users/salex/Documents/GitHub/PictureCrypt/controllerpc.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/controllerpc.cpp

8.3 EncryptDialog Class Reference

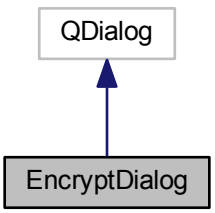
The EncryptDialog class Class to get the image and key to store secret info.

```
#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_horizontalSlider_valueChanged (int value)

  EncryptDialog::on_horizontalSlider_valueChanged Slot if value of the slider is changed. Key is generated here.

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

8.3.1 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.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 EncryptDialog()

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

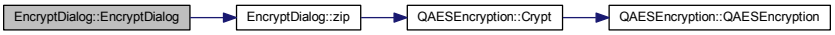
EncryptDialog::EncryptDialog Constructor of the class. Input data is saved here and some variables are set here.

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:



8.3.2.2 ~EncryptDialog()

```
EncryptDialog::~EncryptDialog ( )
```

Definition at line 29 of file encryptdialog.cpp.

8.3.3 Member Function Documentation

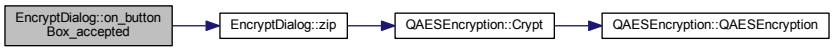
8.3.3.1 on_buttonBox_accepted

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

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

Definition at line 85 of file encryptdialog.cpp.

Here is the call graph for this function:



8.3.3.2 on_buttonBox_rejected

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

EncryptDialog::on_buttonBox_rejected Slot to reject the encryption.

Definition at line 100 of file encryptdialog.cpp.

8.3.3.3 on_fileButton_clicked

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

EncryptDialog::on_fileButton_clicked Slot to select the image.

Definition at line 60 of file encryptdialog.cpp.

8.3.3.4 on_horizontalSlider_valueChanged

```
void EncryptDialog::on_horizontalSlider_valueChanged (
            int value )  [slot]
```

EncryptDialog::on_horizontalSlider_valueChanged Slot if value of the slider is changed. Key is generated here.

Parameters

value	Value of the slider.

Definition at line 110 of file encryptdialog.cpp.

8.3.3.5 zip()

```
QByteArray EncryptDialog::zip ( )
```

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

Returns

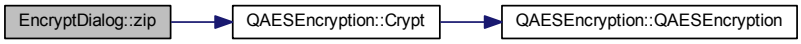
Returns Compressed data.

See also

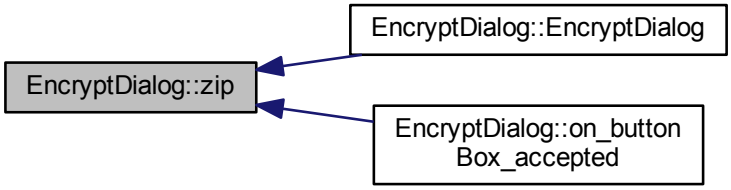
ModelPC::unzip

Definition at line 49 of file encryptdialog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.4 Member Data Documentation

8.3.4.1 bitsUsed

```
int EncryptDialog::bitsUsed
```

bitsUsed Bits used per byte of pixel.

See also

ModelPC::circuit

Definition at line 75 of file encryptdialog.h.

8.3.4.2 compr_data

```
QByteArray EncryptDialog::compr_data
```

compr_data Compressed data, aka Output data.

Definition at line 50 of file encryptdialog.h.

8.3.4.3 data

```
QByteArray EncryptDialog::data
```

data Input data

Definition at line 42 of file encryptdialog.h.

8.3.4.4 goodPercentage

```
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.

8.3.4.5 image

```
QImage EncryptDialog::image
```

image Inputted image

Definition at line 79 of file encryptdialog.h.

8.3.4.6 inputFileName

```
QString EncryptDialog::inputFileName
```

inputFileName Filename of the image.

Definition at line 54 of file encryptdialog.h.

8.3.4.7 key

```
QString EncryptDialog::key
```

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

Definition at line 62 of file encryptdialog.h.

8.3.4.8 size

```
long long int EncryptDialog::size
```

size Size of the image in square pixels

Definition at line 58 of file encryptdialog.h.

8.3.4.9 success

```
bool EncryptDialog::success
```

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

Definition at line 46 of file encryptdialog.h.

8.3.4.10 val

```
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:

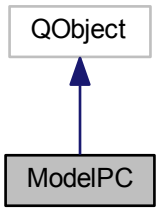
- C:/Users/salex/Documents/GitHub/PictureCrypt/encryptdialog.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/encryptdialog.cpp

8.4 ModelPC Class Reference

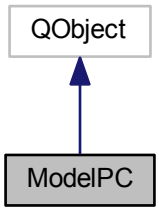
The ModelPC class Model Layer of the app. Controled by ControllerPC.

`#include <modelpc.h>`

Inheritance diagram for ModelPC:



Collaboration diagram for ModelPC:



Public Slots

- void start (QByteArray data, QImage image, int mode=0, QString key="", int _bitsUsed=8)

  ModelPC::start Slot to zip and encrypt data and provide it with some extra stuff After completion start standard ModelPC::encrypt Isn't used in PictureCrypt, but used can be used in other - custom projects.
- void encrypt (QByteArray encr_data, QImage ∗image, int mode=0)

  ModelPC::encrypt Slot to be called when encrypt mode in ViewPC is selected and started.
- void decrypt (QImage ∗image)

  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.

Signals

- alertView (QString message, bool isWarning)

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

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

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

  setProgress Signal to be called to set progress of ProgressDialog.

Public Member Functions

- ModelPC (long _version=131072)

  ModelPC::ModelPC Constructor.
- QByteArray unzip (QByteArray data, QByteArray key)

  ModelPC::unzip Unzip data from ModelPC::decrypt. Just mirrored EncryptDialog::zip.
- void alert (QString message, bool isWarning=false)

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

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 app
- int curMode

  curMode Mode of en- or decryption
- int bitsUsed

  bitsUsed Bits per byte used in pixel
- 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.
- QByteArray zip (QByteArray data, QByteArray key)

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

8.4.1 Detailed Description

The ModelPC class Model Layer of the app. Controled by ControllerPC.

See also

ViewPC, ControllerPC

Definition at line 26 of file modelpc.h.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 ModelPC()

```
ModelPC::ModelPC (
            long _version = 131072 )
```

ModelPC::ModelPC Constructor.

Parameters

_version	Version of the app from Controller.

See also

ControllerPC, ViewPC

Definition at line 8 of file modelpc.cpp.

8.4.3 Member Function Documentation

8.4.3.1 alert()

```
void ModelPC::alert (
            QString message,
            bool isWarning = false )
```

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

Parameters

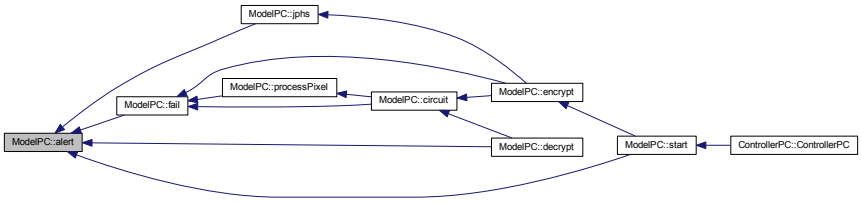
message	Message to be transmitted.
isWarning	Flag if message is critical.

See also

ViewPC::alert

Definition at line 530 of file modelpc.cpp.

Here is the caller graph for this function:



8.4.3.2 alertView

```
ModelPC::alertView (
            QString message,
            bool isWarning )  [signal]
```

alertView Signal to be called to create MessageBox.

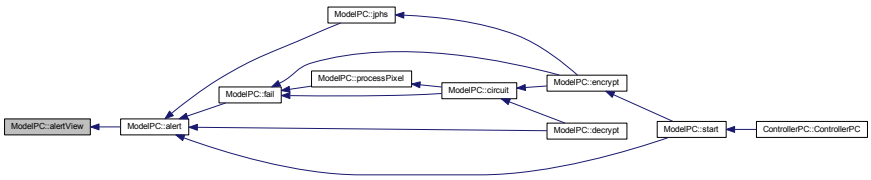
Parameters

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

See also

ModelPC::alert, ViewPC::alert

Here is the caller graph for this function:



8.4.3.3 circuit()

```
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

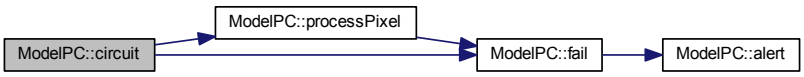
image	Image to be processed.
data	Data to be processed.
countBytes	Number of bytes to be read or written.

See also

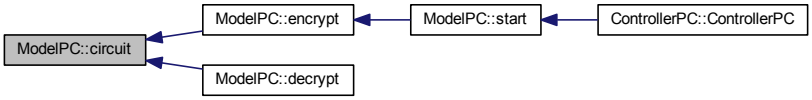
ModelPC::processPixel

Definition at line 234 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.4 decrypt

```
void ModelPC::decrypt (
            QImage * image )  [slot]
```

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

Parameters

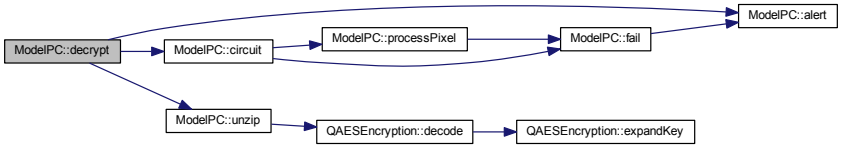
image	Image to be decrypted.

See also

ViewPC::on_startButton_clicked, ModelPC::encrypt, ModelPC::circuit

Definition at line 92 of file modelpc.cpp.

Here is the call graph for this function:



8.4.3.5 encrypt

```
void ModelPC::encrypt (
            QByteArray encr_data,
            QImage * image,
            int mode = 0 )  [slot]
```

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

Parameters

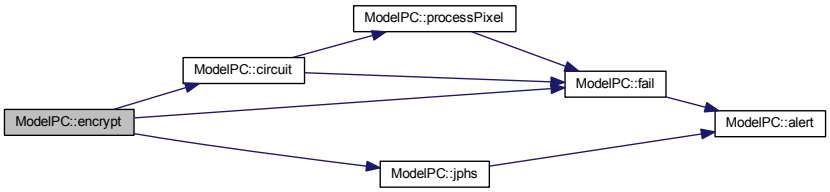
encr_data	Data to be inserted to an image.
image	Image to be inserted in.
mode	Mode of encryption

See also

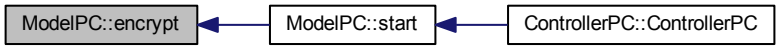
ViewPC::on_startButton_clicked, ModelPC::decrypt, ModelPC::circuit

Definition at line 61 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.6 fail

```
void ModelPC::fail (
            QString message )  [slot]
```

ModelPC::fail Slot to stop execution of cryption.

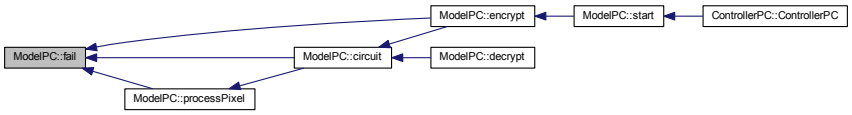
Parameters

message	Message for user

Definition at line 161 of file modelpc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



8.4.3.7 jphs()

```
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 173 of file modelpc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.4.3.8 processPixel()

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

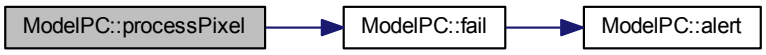
ModelPC::processPixel Processes every pixel. Reads its contains or writes data.

Parameters

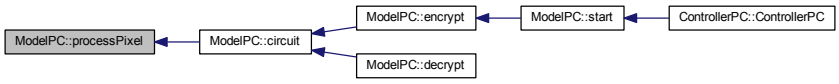
pos	Position of pixel
were	Vector array containing pixels, that were already processed.
isEncrypt	Mode of operation. If true encryption operations will continue, else the decryption ones.

Definition at line 376 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.9 saveData

```
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:

8.4.3.10 saveImage

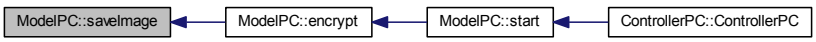
```
ModelPC::saveImage (
            QImage * image )  [signal]
```

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:



8.4.3.11 setProgress

```
ModelPC::setProgress (
            int val )  [signal]
```

setProgress Signal to be called to set progress of ProgressDialog.

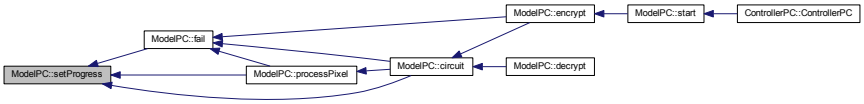
Parameters

val	Value to be set.

See also

ViewPC::setProgress

Here is the caller graph for this function:



8.4.3.12 start

```
void ModelPC::start (
            QByteArray data,
            QImage image,
            int _bitsUsed = 0,
            QString key = "",
            int mode = 8 )  [slot]
```

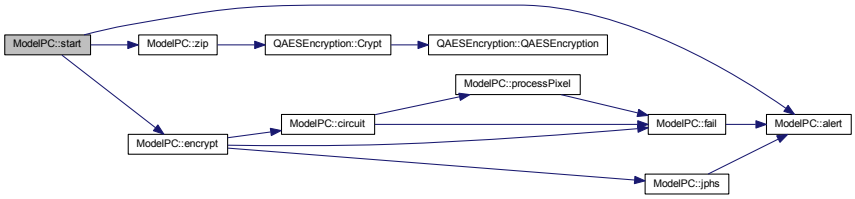
ModelPC::start Slot to zip and encrypt data and provide it with some extra stuff After completion start standard ModelPC::encrypt Isn't used in PictureCrypt, but used can be used in other - custom projects.

Parameters

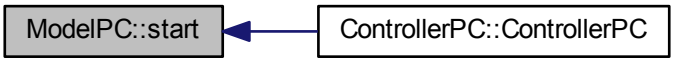
data	Data to be encrypted
image	Image to be encrypted into.
_bitsUsed	Bits per byte, see also ModelPC::bitsUsed
key	Key, if default (empty), random key of 64 charachters will be generated.
mode	Mode of encryption

Definition at line 26 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.13 unzip()

```
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

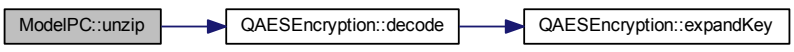
Returns data

See also

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

Definition at line 469 of file modelpc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

8.4.3.14 zip()

```
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::encrypt, ModelPC::unzip

Definition at line 486 of file modelpc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.4.4 Member Data Documentation

8.4.4.1 bitsUsed

```
int ModelPC::bitsUsed
```

bitsUsed Bits per byte used in pixel

Definition at line 79 of file modelpc.h.

8.4.4.2 curMode

```
int ModelPC::curMode
```

curMode Mode of en- or decryption

Definition at line 75 of file modelpc.h.

8.4.4.3 defaultJPHSDir

```
QString ModelPC::defaultJPHSDir
```

defaultJPHSDir Default JPHS directory

Definition at line 83 of file modelpc.h.

8.4.4.4 success

```
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 67 of file modelpc.h.

8.4.4.5 version

```
long ModelPC::version
```

version Version of the app

Definition at line 71 of file modelpc.h.

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

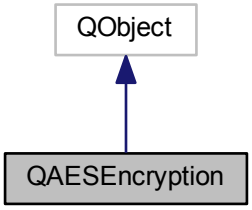
- C:/Users/salex/Documents/GitHub/PictureCrypt/modelpc.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/modelpc.cpp

8.5 QAESEncryption Class Reference

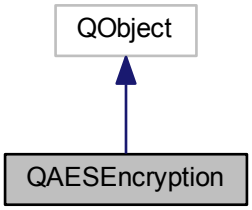
The QAESEncryption class Small and portable AES encryption class for Qt. Supports all key sizes - 128/192/256 bits - ECB, CBC, CFB and OFB modes. Class made entirely by bricke. Github: https://github.com/bricke/Qt-AES.

```
#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 QByte↩ Array &rawText, const QByteArray &key, const QByteArray &iv=NULL, QAESEncryption::Padding padding=QAESEncryption::ISO)

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

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

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

  RemovePadding Removes padding.

8.5.1 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.

8.5.2 Member Enumeration Documentation

8.5.2.1 Aes

`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.

8.5.2.2 Mode

`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.

8.5.2.3 Padding

`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.

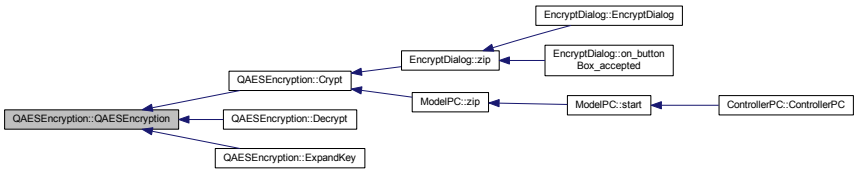
8.5.3 Constructor & Destructor Documentation

8.5.3.1 QAESEncryption()

```
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:



8.5.4 Member Function Documentation

8.5.4.1 Crypt()

```
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

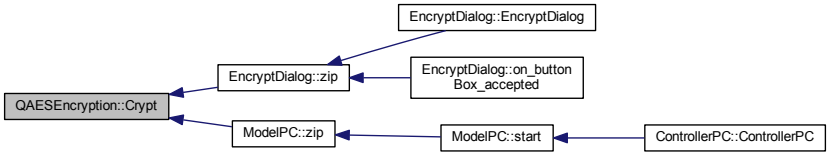
See also

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:



8.5.4.2 decode()

```
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

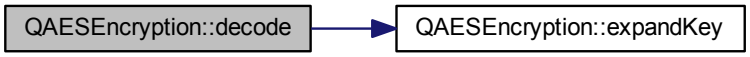
Returns decoded data

See also

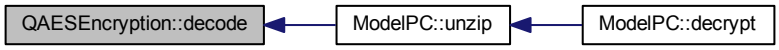
QAESEncryption::Decrypt, QAESEncryption::encode

Definition at line 441 of file qaesencryption.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.4.3 Decrypt()

```
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

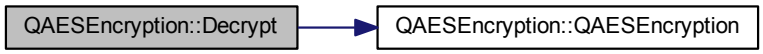
Returns Decrypted data

See also

QAESEncryption::decode, QAESEncryption::Crypt

Definition at line 12 of file qaesencryption.cpp.

Here is the call graph for this function:



8.5.4.4 encode()

```
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

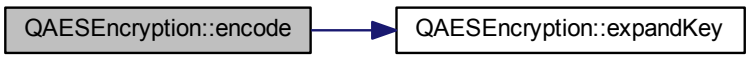
Returns encoded data

See also

QAESEncryption::Crypt, QAESEncryption::decode

Definition at line 391 of file qaesencryption.cpp.

Here is the call graph for this function:



8.5.4.5 ExpandKey()

```
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

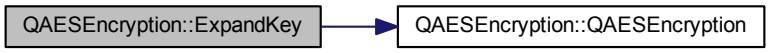
Returns expanded key (I guess)

See also

QAESEncryption::expandKey

Definition at line 18 of file qaesencryption.cpp.

Here is the call graph for this function:



8.5.4.6 expandKey()

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

ExpandKey Expands the key.

Note

Basically the non-static method of QAESEncryption::ExpandKey

Parameters

key	key

Returns

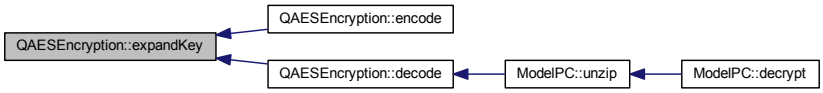
Returns expanded key (I guess)

See also

QAESEncryption::ExpandKey

Definition at line 132 of file qaesencryption.cpp.

Here is the caller graph for this function:



8.5.4.7 RemovePadding()

```
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.

8.5.4.8 removePadding()

```
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:

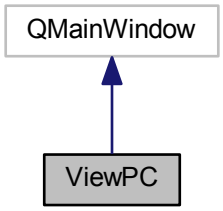
- C:/Users/salex/Documents/GitHub/PictureCrypt/aes/qaesencryption.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/aes/qaesencryption.cpp

8.6 ViewPC Class Reference

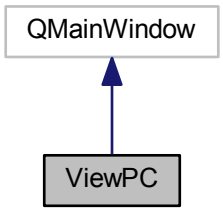
The ViewPC class View layer of the app. Controls EncryptDialog and ProgressDialog.

`#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

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

  encrypt Signal calling ModelPC::encrypt

- decrypt (QImage ∗_image)

  decrypt Signal calling ModelPC::decrypt

- abortModel ()

  abortModel Signal calling to stop ModelPC::circuit

- setBitsUsed (int bitsUsed)

  setBitsUsed Sets bits used in ModelPC

- setJPHSDir (QString dir)

  setJPHSPath Sets the default JPHS directory

Public Member Functions

- ViewPC (QWidget ∗parent=0)
- ∼ViewPC ()

Public Attributes

- QProgressDialog ∗ dialog

  dialog ProgressDialog used.

- bool progressDialogClosed

  progressDialogClosed Flag, if dialog is closed.

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.

8.6.1 Detailed Description

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

See also

ControllerPC, ModelPC, EncryptDialog

Definition at line 29 of file viewpc.h.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 ViewPC()

```
ViewPC::ViewPC (
            QWidget * parent = 0 )  [explicit]
```

Definition at line 6 of file viewpc.cpp.

8.6.2.2 ~ViewPC()

```
ViewPC::~ViewPC ( )
```

Definition at line 13 of file viewpc.cpp.

8.6.3 Member Function Documentation

8.6.3.1 abortCircuit

```
void ViewPC::abortCircuit ( )  [slot]
```

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

Definition at line 203 of file viewpc.cpp.

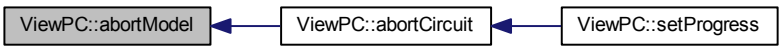
Here is the caller graph for this function:

8.6.3.2 abortModel

```
ViewPC::abortModel ( )  [signal]
```

abortModel Signal calling to stop ModelPC::circuit

Here is the caller graph for this function:



8.6.3.3 alert

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

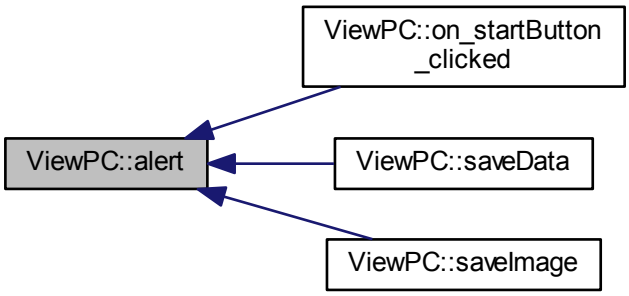
ViewPC::alert Slot to create QMessageBox with message.

Parameters

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

Definition at line 120 of file viewpc.cpp.

Here is the caller graph for this function:



8.6.3.4 decrypt

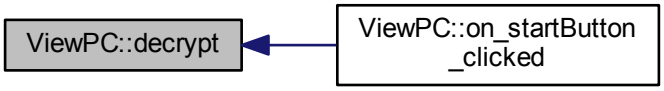
```
ViewPC::decrypt (
            QImage * _image )  [signal]
```

decrypt Signal calling ModelPC::decrypt

Parameters

_image	Image for decryption

Here is the caller graph for this function:



8.6.3.5 encrypt

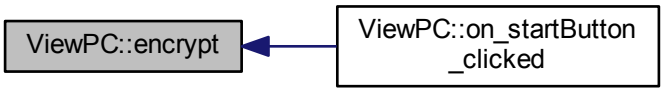
```
ViewPC::encrypt (
            QByteArray data,
            QImage * image,
            int mode )  [signal]
```

encrypt Signal calling ModelPC::encrypt

Parameters

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

Here is the caller graph for this function:



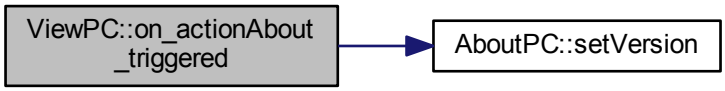
8.6.3.6 on_actionAbout_triggered

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

ViewPC::on_actionAbout_triggered Opens about page.

Definition at line 238 of file viewpc.cpp.

Here is the call graph for this function:



8.6.3.7 on_actionHelp_triggered

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

ViewPC::on_actionHelp_triggered Opens online documentation.

Definition at line 248 of file viewpc.cpp.

8.6.3.8 on_fileButton_clicked

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

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

Definition at line 32 of file viewpc.cpp.

8.6.3.9 on_startButton_clicked

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

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

8.6.4 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

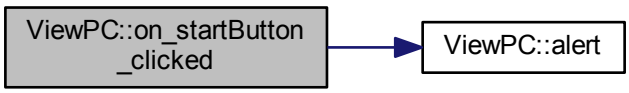
Then the EncryptDialog opens and image and key is selected. Then the ViewPC::encrypt signal is called to start ModelPC::encrypt

8.6.5 Decrypting

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

Definition at line 54 of file viewpc.cpp.

Here is the call graph for this function:



8.6.5.1 saveData

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

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

Parameters

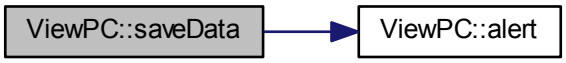
Edata	Encrypted data to be saved.

See also

ModelPC::encrypt

Definition at line 138 of file viewpc.cpp.

Here is the call graph for this function:



8.6.5.2 saveImage

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

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

Parameters

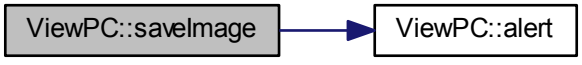
image	Image to be saved.

See also

ModelPC::decrypt

Definition at line 159 of file viewpc.cpp.

Here is the call graph for this function:



8.6.5.3 setBitsUsed

```
ViewPC::setBitsUsed (
            int bitsUsed )  [signal]
```

setBitsUsed Sets bits used in ModelPC

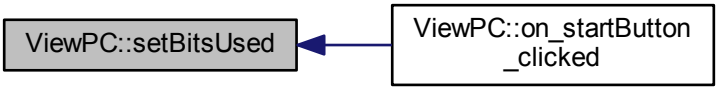
Parameters

bitsUsed	The new value

See also

ModelPC::bitsUsed

Here is the caller graph for this function:



8.6.5.4 setEncryptMode

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

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

Parameters

encr	

Definition at line 216 of file viewpc.cpp.

8.6.5.5 setJPHSDir

```
ViewPC::setJPHSDir (
            QString dir )  [signal]
```

setJPHSPath Sets the default JPHS directory

Parameters

dir	Directory

8.6.5.6 setProgress

```
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 177 of file viewpc.cpp.

Here is the call graph for this function:

8.6.5.7 setVersion

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

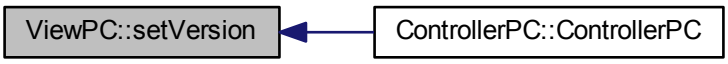
ViewPC::setVersion Set the version of the app from ControllerPC.

Parameters

version	Version as QString

Definition at line 225 of file viewpc.cpp.

Here is the caller graph for this function:



8.6.6 Member Data Documentation

8.6.6.1 dialog

```
QProgressDialog* ViewPC::dialog
```

dialog ProgressDialog used.

See also

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

Definition at line 92 of file viewpc.h.

8.6.6.2 progressDialogClosed

```
bool ViewPC::progressDialogClosed
```

progressDialogClosed Flag, if dialog is closed.

See also

ViewPC::abortCircuit, ViewPC::setProgress

Definition at line 97 of file viewpc.h.

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

- C:/Users/salex/Documents/GitHub/PictureCrypt/viewpc.h
- C:/Users/salex/Documents/GitHub/PictureCrypt/viewpc.cpp

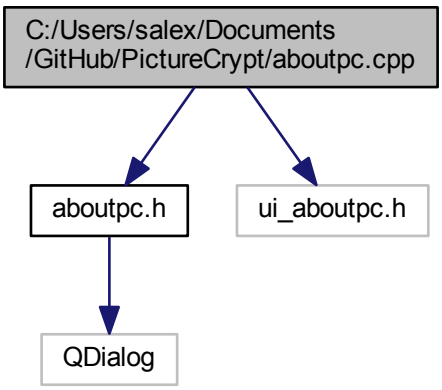
Chapter 9

File Documentation

9.1 C:/Users/salex/Documents/GitHub/PictureCrypt/aboutpc.cpp File Reference

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

Include dependency graph for aboutpc.cpp:



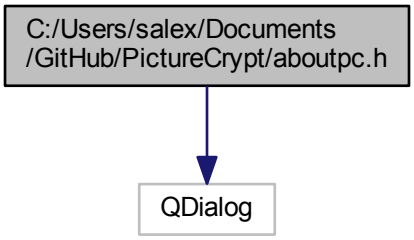
9.2 aboutpc.cpp

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

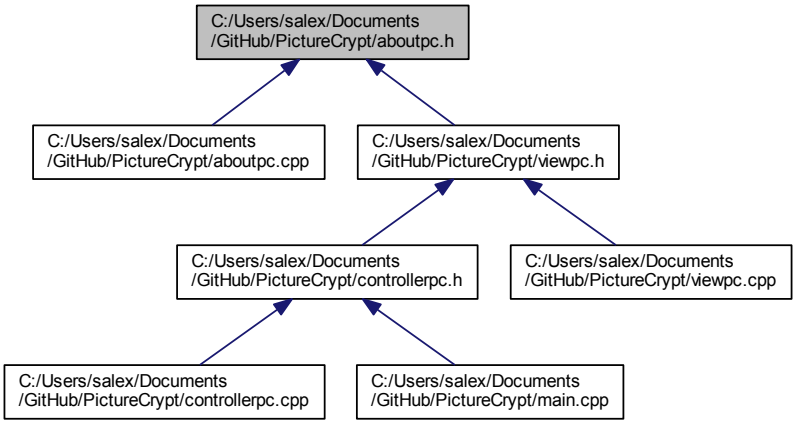
9.3 C:/Users/salex/Documents/GitHub/PictureCrypt/aboutpc.h File Reference

```
#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

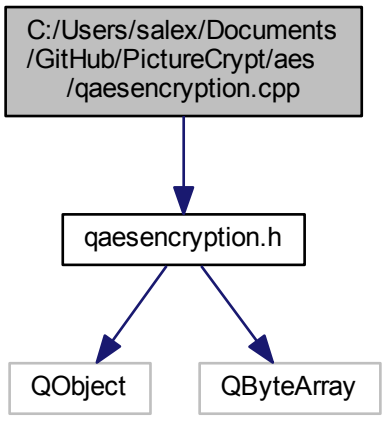
- Ui

9.4 aboutpc.h

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

9.5 C:/Users/salex/Documents/GitHub/PictureCrypt/aes/qaesencryption.cpp File Reference

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



Functions

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

9.5.1 Function Documentation

9.5.1.1 multiply()

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

Definition at line 57 of file qaesencryption.cpp.

Here is the call graph for this function:

9.5.1.2 xTime()

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

Definition at line 53 of file qaesencryption.cpp.

Here is the caller graph for this function:

9.6 qaesencryption.cpp

```
00001 #include "qaesencryption.h"
00002
00003 /*
00004  * Static Functions
00005  * */
00006 QByteArray QAESEncryption::Crypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText,
00007                                 const QByteArray &key, const QByteArray &iv,
      QAESEncryption::Padding padding)
00008 {
00009     return QAESEncryption(level, mode, padding).encode(rawText, key, iv);
00010 }
00011
```

```
00012 QByteArray QAESEncryption::Decrypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText,
00013                                    const QByteArray &key, const QByteArray &iv,
      QAESEncryption::Padding padding)
00014 {
00015     return QAESEncryption(level, mode, padding).decode(rawText, key, iv);
00016 }
00017
00018 QByteArray QAESEncryption::ExpandKey(
      QAESEncryption::Aes level, QAESEncryption::Mode mode, const
      QByteArray &key)
00019 {
00020     return QAESEncryption(level, mode).expandKey(key);
00021 }
00022
00023 QByteArray QAESEncryption::RemovePadding(const QByteArray &rawText,
      QAESEncryption::Padding padding)
00024 {
00025     QByteArray ret(rawText);
00026     switch (padding)
00027     {
00028     case Padding::ZERO:
00029         //Works only if the last byte of the decoded array is not zero
00030         while (ret.at(ret.length()-1) == 0x00)
00031             ret.remove(ret.length()-1, 1);
00032         break;
00033     case Padding::PKCS7:
00034         ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00035         break;
00036     case Padding::ISO:
00037         ret.truncate(ret.lastIndexOf(0x80));
00038         break;
00039     default:
00040         //do nothing
00041         break;
00042     }
00043     return ret;
00044 }
00045 /*
00046  * End Static function declarations
00047  * */
00048
00049 /*
00050  * Inline Functions
00051  * */
00052
00053 inline quint8 xTime(quint8 x){
00054   return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
00055 }
00056
00057 inline quint8 multiply(quint8 x, quint8 y){
00058   return (((y & 1) * x) ^ ((y>>1 & 1) * xTime(x)) ^ ((y>>2 & 1) * xTime(
      xTime(x))) ^ ((y>>3 & 1)
00059             * xTime(xTime(xTime(x)))) ^ ((y>>4 & 1) * xTime(
      xTime(xTime(xTime(x))))));
00060 }
00061
00062 /*
00063  * End Inline functions
00064  * */
00065
00066
00067 QAESEncryption::QAESEncryption(Aes level, Mode mode,
00068                                Padding padding)
00069     : m_nb(4), m_blocklen(16), m_level(level), m_mode(mode), m_padding(padding)
00070 {
00071     m_state = NULL;
00072
00073     switch (level)
00074     {
00075     case AES_128: {
00076         AES128 aes;
00077         m_nk = aes.nk;
00078         m_keyLen = aes.keylen;
00079         m_nr = aes.nr;
00080         m_expandedKey = aes.expandedKey;
00081         }
00082         break;
00083     case AES_192: {
00084         AES192 aes;
00085         m_nk = aes.nk;
00086         m_keyLen = aes.keylen;
00087         m_nr = aes.nr;
00088         m_expandedKey = aes.expandedKey;
00089         }
00090         break;
00091     case AES_256: {
```

```
00092         AES256 aes;
00093         m_nk = aes.nk;
00094         m_keyLen = aes.keylen;
00095         m_nr = aes.nr;
00096         m_expandedKey = aes.expandedKey;
00097         }
00098         break;
00099     default: {
00100         AES128 aes;
00101         m_nk = aes.nk;
00102         m_keyLen = aes.keylen;
00103         m_nr = aes.nr;
00104         m_expandedKey = aes.expandedKey;
00105         }
00106         break;
00107     }
00108
00109 }
00110 QByteArray QAESEncryption::getPadding(int currSize, int alignment)
00111 {
00112     int size = (alignment - currSize % alignment) % alignment;
00113     if (size == 0) return QByteArray();
00114     switch(m_padding)
00115     {
00116     case Padding::ZERO:
00117         return QByteArray(size, 0x00);
00118         break;
00119     case Padding::PKCS7:
00120         return QByteArray(size,size);
00121         break;
00122     case Padding::ISO:
00123         return QByteArray (size-1, 0x00).prepend(0x80);
00124         break;
00125     default:
00126         return QByteArray(size, 0x00);
00127         break;
00128     }
00129     return QByteArray(size, 0x00);
00130 }
00131
00132 QByteArray QAESEncryption::expandKey(const QByteArray &key)
00133 {
00134   int i, k;
00135   quint8 tempa[4]; // Used for the column/row operations
00136   QByteArray roundKey(key);
00137
00138   // The first round key is the key itself.
00139   // ...
00140
00141   // All other round keys are found from the previous round keys.
00142   //i == Nk
00143   for(i = m_nk; i < m_nb * (m_nr + 1); i++)
00144   {
00145     tempa[0] = (quint8) roundKey.at((i-1) * 4 + 0);
00146     tempa[1] = (quint8) roundKey.at((i-1) * 4 + 1);
00147     tempa[2] = (quint8) roundKey.at((i-1) * 4 + 2);
00148     tempa[3] = (quint8) roundKey.at((i-1) * 4 + 3);
00149
00150     if (i % m_nk == 0)
00151     {
00152         // This function shifts the 4 bytes in a word to the left once.
00153         // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]
00154
00155         // Function RotWord()
00156         k = tempa[0];
00157         tempa[0] = tempa[1];
00158         tempa[1] = tempa[2];
00159         tempa[2] = tempa[3];
00160         tempa[3] = k;
00161
00162         // Function Subword()
00163         tempa[0] = getSBoxValue(tempa[0]);
00164         tempa[1] = getSBoxValue(tempa[1]);
00165         tempa[2] = getSBoxValue(tempa[2]);
00166         tempa[3] = getSBoxValue(tempa[3]);
00167
00168         tempa[0] =  tempa[0] ^ Rcon[i/m_nk];
00169     }
00170     if (m_level == AES_256 && i % m_nk == 4)
00171     {
00172         // Function Subword()
00173         tempa[0] = getSBoxValue(tempa[0]);
00174         tempa[1] = getSBoxValue(tempa[1]);
00175         tempa[2] = getSBoxValue(tempa[2]);
00176         tempa[3] = getSBoxValue(tempa[3]);
00177     }
00178     roundKey.insert(i * 4 + 0, (quint8) roundKey.at((i - m_nk) * 4 + 0) ^ tempa[0]);
```

```
00179     roundKey.insert(i * 4 + 1, (quint8) roundKey.at((i - m_nk) * 4 + 1) ^ tempa[1]);
00180     roundKey.insert(i * 4 + 2, (quint8) roundKey.at((i - m_nk) * 4 + 2) ^ tempa[2]);
00181     roundKey.insert(i * 4 + 3, (quint8) roundKey.at((i - m_nk) * 4 + 3) ^ tempa[3]);
00182   }
00183   return roundKey;
00184 }
00185
00186 // This function adds the round key to state.
00187 // The round key is added to the state by an XOR function.
00188 void QAESEncryption::addRoundKey(const quint8 round, const QByteArray expKey)
00189 {
00190   QByteArray::iterator it = m_state->begin();
00191   for(int i=0; i < 16; ++i)
00192       it[i] = (quint8) it[i] ^ (quint8) expKey.at(round * m_nb * 4 + (i/4) * m_nb + (i%4));
00193 }
00194
00195 // The SubBytes Function Substitutes the values in the
00196 // state matrix with values in an S-box.
00197 void QAESEncryption::subBytes()
00198 {
00199   QByteArray::iterator it = m_state->begin();
00200   for(int i = 0; i < 16; i++)
00201     it[i] = getSBoxValue((quint8) it[i]);
00202 }
00203
00204 // The ShiftRows() function shifts the rows in the state to the left.
00205 // Each row is shifted with different offset.
00206 // Offset = Row number. So the first row is not shifted.
00207 void QAESEncryption::shiftRows()
00208 {
00209     QByteArray::iterator it = m_state->begin();
00210     quint8 temp;
00211     //Keep in mind that QByteArray is column-driven!!
00212
00213      //Shift 1 to left
00214     temp   = (quint8)it[1];
00215     it[1]  = (quint8)it[5];
00216     it[5]  = (quint8)it[9];
00217     it[9]  = (quint8)it[13];
00218     it[13] = (quint8)temp;
00219
00220     //Shift 2 to left
00221     temp   = (quint8)it[2];
00222     it[2]  = (quint8)it[10];
00223     it[10] = (quint8)temp;
00224     temp   = (quint8)it[6];
00225     it[6]  = (quint8)it[14];
00226     it[14] = (quint8)temp;
00227
00228     //Shift 3 to left
00229     temp   = (quint8)it[3];
00230     it[3]  = (quint8)it[15];
00231     it[15] = (quint8)it[11];
00232     it[11] = (quint8)it[7];
00233     it[7]  = (quint8)temp;
00234 }
00235
00236 // MixColumns function mixes the columns of the state matrix
00237 //optimized!!
00238 void QAESEncryption::mixColumns()
00239 {
00240   QByteArray::iterator it = m_state->begin();
00241   quint8 tmp, tm, t;
00242
00243   for(int i = 0; i < 16; i += 4){
00244     t       = (quint8)it[i];
00245     tmp     =  (quint8)it[i] ^ (quint8)it[i+1] ^ (quint8)it[i+2] ^ (quint8)it[i+3] ;
00246
00247     tm      = xTime( (quint8)it[i] ^ (quint8)it[i+1] );
00248     it[i]   = (quint8)it[i] ^ (quint8)tm ^ (quint8)tmp;
00249
00250     tm      = xTime( (quint8)it[i+1] ^ (quint8)it[i+2]);
00251     it[i+1] = (quint8)it[i+1] ^ (quint8)tm ^ (quint8)tmp;
00252
00253     tm      = xTime( (quint8)it[i+2] ^ (quint8)it[i+3]);
00254     it[i+2] =(quint8)it[i+2] ^ (quint8)tm ^ (quint8)tmp;
00255
00256     tm      = xTime((quint8)it[i+3] ^ (quint8)t);
00257     it[i+3] =(quint8)it[i+3] ^ (quint8)tm ^ (quint8)tmp;
00258   }
00259 }
00260
00261 // MixColumns function mixes the columns of the state matrix.
00262 // The method used to multiply may be difficult to understand for the inexperienced.
00263 // Please use the references to gain more information.
00264 void QAESEncryption::invMixColumns()
00265 {
```

```
00266   QByteArray::iterator it = m_state->begin();
00267   quint8 a,b,c,d;
00268   for(int i = 0; i < 16; i+=4){
00269     a = (quint8) it[i];
00270     b = (quint8) it[i+1];
00271     c = (quint8) it[i+2];
00272     d = (quint8) it[i+3];
00273
00274     it[i]   = (quint8) (multiply(a, 0x0e) ^ multiply(b, 0x0b) ^
      multiply(c, 0x0d) ^ multiply(d, 0x09));
00275     it[i+1] = (quint8) (multiply(a, 0x09) ^ multiply(b, 0x0e) ^
      multiply(c, 0x0b) ^ multiply(d, 0x0d));
00276     it[i+2] = (quint8) (multiply(a, 0x0d) ^ multiply(b, 0x09) ^
      multiply(c, 0x0e) ^ multiply(d, 0x0b));
00277     it[i+3] = (quint8) (multiply(a, 0x0b) ^ multiply(b, 0x0d) ^
      multiply(c, 0x09) ^ multiply(d, 0x0e));
00278   }
00279 }
00280
00281 // The SubBytes Function Substitutes the values in the
00282 // state matrix with values in an S-box.
00283 void QAESEncryption::invSubBytes()
00284 {
00285     QByteArray::iterator it = m_state->begin();
00286     for(int i = 0; i < 16; ++i)
00287         it[i] = getSBoxInvert((quint8) it[i]);
00288 }
00289
00290 void QAESEncryption::invShiftRows()
00291 {
00292     QByteArray::iterator it = m_state->begin();
00293     uint8_t temp;
00294
00295     //Keep in mind that QByteArray is column-driven!!
00296
00297     //Shift 1 to right
00298     temp   = (quint8)it[13];
00299     it[13] = (quint8)it[9];
00300     it[9]  = (quint8)it[5];
00301     it[5]  = (quint8)it[1];
00302     it[1]  = (quint8)temp;
00303
00304     //Shift 2
00305     temp   = (quint8)it[10];
00306     it[10] = (quint8)it[2];
00307     it[2]  = (quint8)temp;
00308     temp   = (quint8)it[14];
00309     it[14] = (quint8)it[6];
00310     it[6]  = (quint8)temp;
00311
00312     //Shift 3
00313     temp   = (quint8)it[15];
00314     it[15] = (quint8)it[3];
00315     it[3]  = (quint8)it[7];
00316     it[7]  = (quint8)it[11];
00317     it[11] = (quint8)temp;
00318 }
00319
00320 QByteArray QAESEncryption::byteXor(const QByteArray &a, const QByteArray &b)
00321 {
00322   QByteArray::const_iterator it_a = a.begin();
00323   QByteArray::const_iterator it_b = b.begin();
00324   QByteArray ret;
00325
00326   //for(int i = 0; i < m_blocklen; i++)
00327   for(int i = 0; i < std::min(a.size(), b.size()); i++)
00328       ret.insert(i,it_a[i] ^ it_b[i]);
00329
00330   return ret;
00331 }
00332
00333 // Cipher is the main function that encrypts the PlainText.
00334 QByteArray QAESEncryption::cipher(const QByteArray &expKey, const QByteArray &in)
00335 {
00336
00337   //m_state is the input buffer...
00338   QByteArray output(in);
00339   m_state = &output;
00340
00341   // Add the First round key to the state before starting the rounds.
00342   addRoundKey(0, expKey);
00343
00344   // There will be Nr rounds.
00345   // The first Nr-1 rounds are identical.
00346   // These Nr-1 rounds are executed in the loop below.
00347   for(quint8 round = 1; round < m_nr; ++round){
00348     subBytes();
```

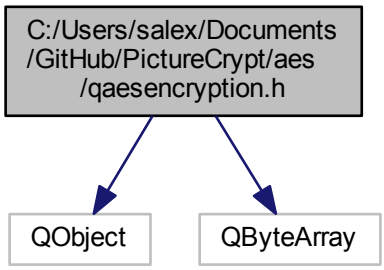
```
00349     shiftRows();
00350     mixColumns();
00351     addRoundKey(round, expKey);
00352   }
00353
00354   // The last round is given below.
00355   // The MixColumns function is not here in the last round.
00356   subBytes();
00357   shiftRows();
00358   addRoundKey(m_nr, expKey);
00359
00360   return output;
00361 }
00362
00363 QByteArray QAESEncryption::invCipher(const QByteArray &expKey, const QByteArray &in)
00364 {
00365     //m_state is the input buffer.... handle it!
00366     QByteArray output(in);
00367     m_state = &output;
00368
00369     // Add the First round key to the state before starting the rounds.
00370     addRoundKey(m_nr, expKey);
00371
00372     // There will be Nr rounds.
00373     // The first Nr-1 rounds are identical.
00374     // These Nr-1 rounds are executed in the loop below.
00375     for(quint8 round=m_nr-1; round>0 ; round--){
00376         invShiftRows();
00377         invSubBytes();
00378         addRoundKey(round, expKey);
00379         invMixColumns();
00380     }
00381
00382     // The last round is given below.
00383     // The MixColumns function is not here in the last round.
00384     invShiftRows();
00385     invSubBytes();
00386     addRoundKey(0, expKey);
00387
00388     return output;
00389 }
00390
00391 QByteArray QAESEncryption::encode(const QByteArray &rawText, const QByteArray &key,
      const QByteArray &iv)
00392 {
00393     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00394        return QByteArray();
00395
00396     QByteArray ret;
00397     QByteArray expandedKey = expandKey(key);
00398     QByteArray alignedText(rawText);
00399
00400     //Fill array with padding
00401     alignedText.append(getPadding(rawText.size(), m_blocklen));
00402
00403     switch(m_mode)
00404     {
00405     case ECB:
00406         for(int i=0; i < alignedText.size(); i+= m_blocklen)
00407             ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00408         break;
00409     case CBC: {
00410             QByteArray ivTemp(iv);
00411             for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00412                 alignedText.replace(i, m_blocklen, byteXor(alignedText.mid(i, m_blocklen),ivTemp));
00413                 ret.append(cipher(expandedKey, alignedText.mid(i, m_blocklen)));
00414                 ivTemp = ret.mid(i, m_blocklen);
00415             }
00416         }
00417         break;
00418     case CFB: {
00419             ret.append(byteXor(alignedText.left(m_blocklen), cipher(expandedKey, iv)));
00420             for(int i=0; i < alignedText.size(); i+= m_blocklen) {
00421                 if (i+m_blocklen < alignedText.size())
00422                     ret.append(byteXor(alignedText.mid(i+m_blocklen, m_blocklen),
00423                                        cipher(expandedKey, ret.mid(i, m_blocklen))));
00424             }
00425         }
00426         break;
00427     case OFB: {
00428             QByteArray ofbTemp;
00429             ofbTemp.append(cipher(expandedKey, iv));
00430             for (int i=m_blocklen; i < alignedText.size(); i += m_blocklen){
00431                 ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00432             }
00433             ret.append(byteXor(alignedText, ofbTemp));
00434         }
```

```
00435         break;
00436     default: break;
00437     }
00438     return ret;
00439 }
00440
00441 QByteArray QAESEncryption::decode(const QByteArray &rawText, const QByteArray &key,
      const QByteArray &iv)
00442 {
00443     if (m_mode >= CBC && (iv.isNull() || iv.size() != m_blocklen))
00444        return QByteArray();
00445
00446     QByteArray ret;
00447     QByteArray expandedKey = expandKey(key);
00448
00449     switch(m_mode)
00450     {
00451     case ECB:
00452         for(int i=0; i < rawText.size(); i+= m_blocklen)
00453             ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00454         break;
00455     case CBC: {
00456             QByteArray ivTemp(iv);
00457             for(int i=0; i < rawText.size(); i+= m_blocklen){
00458                 ret.append(invCipher(expandedKey, rawText.mid(i, m_blocklen)));
00459                 ret.replace(i, m_blocklen, byteXor(ret.mid(i, m_blocklen),ivTemp));
00460                 ivTemp = rawText.mid(i, m_blocklen);
00461             }
00462         }
00463         break;
00464     case CFB: {
00465             ret.append(byteXor(rawText.mid(0, m_blocklen), cipher(expandedKey, iv)));
00466             for(int i=0; i < rawText.size(); i+= m_blocklen){
00467                 if (i+m_blocklen < rawText.size()) {
00468                     ret.append(byteXor(rawText.mid(i+m_blocklen, m_blocklen),
00469                                        cipher(expandedKey, rawText.mid(i, m_blocklen))));
00470                 }
00471             }
00472         }
00473         break;
00474     case OFB: {
00475         QByteArray ofbTemp;
00476         ofbTemp.append(cipher(expandedKey, iv));
00477         for (int i=m_blocklen; i < rawText.size(); i += m_blocklen){
00478             ofbTemp.append(cipher(expandedKey, ofbTemp.right(m_blocklen)));
00479         }
00480         ret.append(byteXor(rawText, ofbTemp));
00481     }
00482         break;
00483     default:
00484         //do nothing
00485         break;
00486     }
00487     return ret;
00488 }
00489
00490 QByteArray QAESEncryption::removePadding(const QByteArray &rawText)
00491 {
00492     QByteArray ret(rawText);
00493     switch (m_padding)
00494     {
00495     case Padding::ZERO:
00496         //Works only if the last byte of the decoded array is not zero
00497         while (ret.at(ret.length()-1) == 0x00)
00498             ret.remove(ret.length()-1, 1);
00499         break;
00500     case Padding::PKCS7:
00501         ret.remove(ret.length() - ret.at(ret.length()-1), ret.at(ret.length()-1));
00502         break;
00503     case Padding::ISO:
00504         ret.truncate(ret.lastIndexOf(0x80));
00505         break;
00506     default:
00507         //do nothing
00508         break;
00509     }
00510     return ret;
00511 }
```

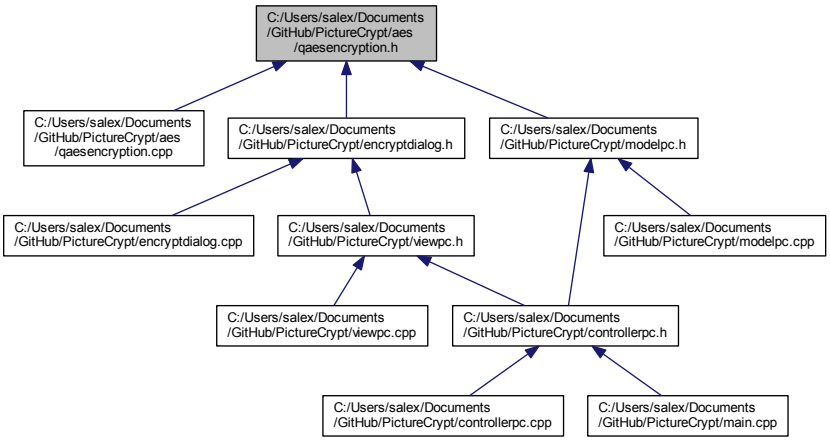
9.7 C:/Users/salex/Documents/GitHub/PictureCrypt/aes/qaesencryption.h File Reference

```
#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.

9.8 qaesencryption.h

```
00001 #ifndef QAESENCRYPTION_H
00002 #define QAESENCRYPTION_H
00003
00004 #include <QObject>
00005 #include <QByteArray>
```

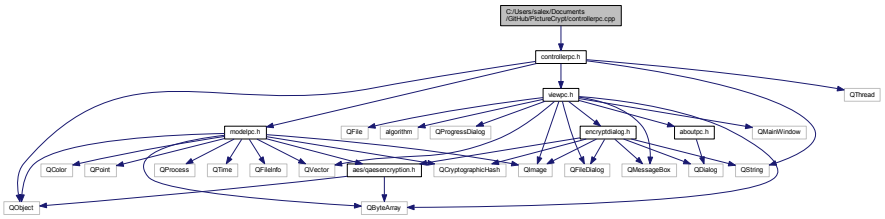
```
00006
00014 class QAESEncryption : public QObject
00015 {
00016     Q_OBJECT
00017 public:
00027     enum Aes {
00028         AES_128,
00029         AES_192,
00030         AES_256
00031     };
00040     enum Mode {
00041         ECB,
00042         CBC,
00043         CFB,
00044         OFB
00045     };
00046
00055     enum Padding {
00056       ZERO,
00057       PKCS7,
00058       ISO
00059     };
00071     static QByteArray Crypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
00072                             const QByteArray &iv = NULL, QAESEncryption::Padding
      padding = QAESEncryption::ISO);
00084     static QByteArray Decrypt(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &rawText, const QByteArray &key,
00085                               const QByteArray &iv = NULL,
      QAESEncryption::Padding padding = QAESEncryption::ISO);
00094     static QByteArray ExpandKey(QAESEncryption::Aes level,
      QAESEncryption::Mode mode, const QByteArray &key);
00102     static QByteArray RemovePadding(const QByteArray &rawText,
      QAESEncryption::Padding padding);
00103
00104     QAESEncryption(QAESEncryption::Aes level,
      QAESEncryption::Mode mode,
00105                    QAESEncryption::Padding padding =
      QAESEncryption::ISO);
00116     QByteArray encode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv = NULL);
00127     QByteArray decode(const QByteArray &rawText, const QByteArray &key, const QByteArray &iv = NULL);
00136     QByteArray removePadding(const QByteArray &rawText);
00145     QByteArray expandKey(const QByteArray &key);
00146
00147 signals:
00148
00149 public slots:
00150
00151 private:
00152     int m_nb;
00153     int m_blocklen;
00154     int m_level;
00155     int m_mode;
00156     int m_nk;
00157     int m_keyLen;
00158     int m_nr;
00159     int m_expandedKey;
00160     int m_padding;
00161     QByteArray* m_state;
00162
00163     struct AES256{
00164         int nk = 8;
00165         int keylen = 32;
00166         int nr = 14;
00167         int expandedKey = 240;
00168     };
00169
00170     struct AES192{
00171         int nk = 6;
00172         int keylen = 24;
00173         int nr = 12;
00174         int expandedKey = 209;
00175     };
00176
00177     struct AES128{
00178         int nk = 4;
00179         int keylen = 16;
00180         int nr = 10;
00181         int expandedKey = 176;
00182     };
00183
00184     quint8 getSBoxValue(quint8 num){return sbox[num];}
00185     quint8 getSBoxInvert(quint8 num){return rsbox[num];}
00186
00187     void addRoundKey(const quint8 round, const QByteArray expKey);
00188     void subBytes();
00189     void shiftRows();
```

```
00190     void mixColumns();
00191     void invMixColumns();
00192     void invSubBytes();
00193     void invShiftRows();
00194     QByteArray getPadding(int currSize, int alignment);
00195     QByteArray cipher(const QByteArray &expKey, const QByteArray &plainText);
00196     QByteArray invCipher(const QByteArray &expKey, const QByteArray &plainText);
00197     QByteArray byteXor(const QByteArray &in, const QByteArray &iv);
00198
00199     const quint8 sbox[256] =   {
00200       //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
00201       0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
00202       0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
00203       0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
00204       0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
00205       0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
00206       0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
00207       0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
00208       0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
00209       0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
00210       0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
00211       0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
00212       0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
00213       0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
00214       0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
00215       0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
00216       0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };
00217
00218     const quint8 rsbox[256] =
00219     { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
00220       0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
00221       0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
00222       0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
00223       0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
00224       0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
00225       0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
00226       0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
00227       0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
00228       0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
00229       0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
00230       0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
00231       0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
00232       0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
00233       0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
00234       0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };
00235
00236     // The round constant word array, Rcon[i], contains the values given by
00237     // x to th e power (i-1) being powers of x (x is denoted as {02}) in the field GF(2^8)
00238     // Only the first 14 elements are needed
00239     const quint8 Rcon[256] = {
00240         0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab/*, 0x4d, 0x9a,
00241         0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
00242         0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
00243         0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
00244         0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
00245         0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
00246         0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
00247         0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
00248         0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
00249         0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
00250         0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
00251         0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
00252         0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
00253         0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
00254         0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
00255             0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
      */};
00256 };
00257
00258 #endif // QAESENCRYPTION_H
```

9.9 C:/Users/salex/Documents/GitHub/PictureCrypt/controllerpc.cpp File Reference

```
#include "controllerpc.h"
```

Include dependency graph for controllerpc.cpp:



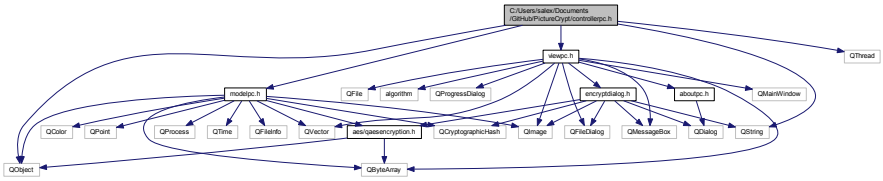
9.10 controllerpc.cpp

```
00001 #include "controllerpc.h"
00002
00009 ControllerPC::ControllerPC()
00010 {
00011     // Version control
00012     QString _version = "1.2.5";
00013     versionString = _version;
00014     auto ver = _version.split(".");
00015     version = ver[0].toInt() * pow(2, 16) + ver[1].toInt() * pow(2, 8) + ver[2].toInt();
00016     // Layer creation
00017     view = new ViewPC();
00018     model = new ModelPC(version);
00019     QThread * modelThread = new QThread();
00020     model->moveToThread(modelThread);
00021     modelThread->start();
00022
00023     view->setVersion(versionString);
00024     view->show();
00025     // Layer Connection
00026     connect(view, SIGNAL(encrypt(QByteArray,QImage*,int)), model, SLOT(encrypt(QByteArray,QImage*,int)));
00027     connect(view, SIGNAL(decrypt(QImage*)), model, SLOT(decrypt(QImage*)));
00028     connect(view, SIGNAL(abortModel()), this, SLOT(abortCircuit()));
00029     connect(view, SIGNAL(setBitsUsed(int)), this, SLOT(setBitsUsed(int)));
00030     connect(view, SIGNAL(setJPHSDir(QString)), this, SLOT(setJPHSDir(QString)));
00031
00032     connect(model, SIGNAL(alertView(QString,bool)), view, SLOT(alert(QString,bool)));
00033     connect(model, SIGNAL(saveData(QByteArray)), view, SLOT(saveData(QByteArray)));
00034     connect(model, SIGNAL(saveImage(QImage*)), view, SLOT(saveImage(QImage*)));
00035     connect(model, SIGNAL(setProgress(int)), view, SLOT(setProgress(int)));
00036 }
00041 void ControllerPC::abortCircuit()
00042 {
00043     model->success = false;
00044 }
00049 void ControllerPC::setBitsUsed(int bitsUsed)
00050 {
00051     model->bitsUsed = bitsUsed;
00052 }
00057 void ControllerPC::setJPHSDir(QString dir)
00058 {
00059     model->defaultJPHSDir = dir;
00060 }
```

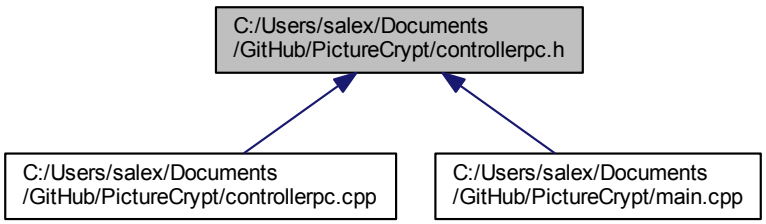
9.11 C:/Users/salex/Documents/GitHub/PictureCrypt/controllerpc.h File Reference

```
#include <QObject>
#include <QString>
#include <QThread>
#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.

9.11.1 Detailed Description

Header of ControllerPC class

See also

ControllerPC, ModelPC, ViewPC

Definition in file controllerpc.h.

9.12 controllerpc.h

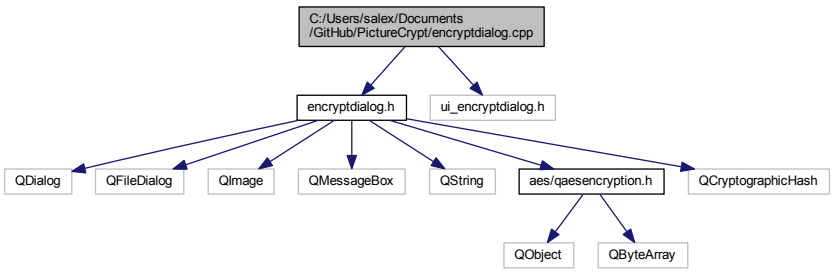
```
00001 #ifndef CONTROLLERPC_H
00002 #define CONTROLLERPC_H
00003
00004 #include <QObject>
00005 #include <QString>
00006 #include <QThread>
00007
00008 #include <modelpc.h>
```

```
00009 #include <viewpc.h>
00019 class ControllerPC : public QObject
00020 {
00021     Q_OBJECT
00022 public:
00023     ControllerPC();
00027     long int version;
00031     QString versionString;
00032 public slots:
00033     void abortCircuit();
00034     void setBitsUsed(int bitsUsed);
00035     void setJPHSDir(QString dir);
00036 private:
00037     ViewPC * view;
00038     ModelPC * model;
00039 };
00040
00041 #endif // CONTROLLERPC_H
```

9.13 C:/Users/salex/Documents/GitHub/PictureCrypt/encryptdialog.cpp File Reference

```
#include "encryptdialog.h"
#include "ui_encryptdialog.h"
```

Include dependency graph for encryptdialog.cpp:



9.14 encryptdialog.cpp

```
00001 #include "encryptdialog.h"
00002 #include "ui_encryptdialog.h"
00009 EncryptDialog::EncryptDialog(QByteArray _data, QWidget *parent) :
00010     QDialog(parent),
00011     ui(new Ui::EncryptDialog)
00012 {
00013     ui->setupUi(this);
00014     data = _data;
00015     success = false;
00016     // UI setup
00017     ui->totalBytes->setText(QString::number(data.size()));
00018     key.clear();
00019     for(int i = 0; i < 24; i++)
00020         key.append(48 + qrand() % 75);
00021     val = 24;
00022     compr_data = zip();
00023     long long int compr_data_size = compr_data.size();
00024     ui->zippedBytes->setText(QString::number(compr_data_size));
00025     goodPercentage = false;
00026     bitsUsed = 8;
00027 }
00028
00029 EncryptDialog::~EncryptDialog()
00030 {
00031     delete ui;
00032 }
00033
```

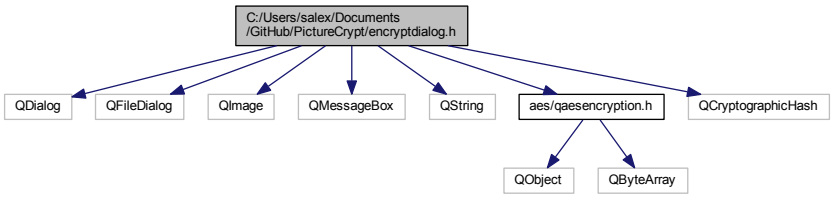
```
00034 void EncryptDialog::alert(QString text)
00035 {
00036     QMessageBox t;
00037     t.setWindowTitle("Message");
00038     t.setIcon(QMessageBox::Warning);
00039     t.setWindowIcon(QIcon(":/mail.png"));
00040     t.setText(text);
00041     t.exec();
00042 }
00049 QByteArray EncryptDialog::zip()
00050 {
00051     // Zip
00052     QByteArray c_data = qCompress(data, 9);
00053     // Encryption
00054     QByteArray hashKey = QCryptographicHash::hash(key.toUtf8(), QCryptographicHash::Sha256);
00055     return QAESEncryption::Crypt(QAESEncryption::AES_256,
      QAESEncryption::ECB, c_data, hashKey);
00056 }
00060 void EncryptDialog::on_fileButton_clicked()
00061 {
00062     // Selet file
00063     inputFileName = QFileDialog::getOpenFileName(this, tr("Open File"), "/", tr("Images (*.png
       *.xpm *.jpg *.jpeg)"));
00064     ui->fileLabel->setText(inputFileName);
00065     // Open image
00066     QImage img(inputFileName);
00067     image = img;
00068     // Get size
00069     size = img.width() * img.height();
00070     // UI setup
00071     long long int compr_data_size = compr_data.size();
00072     ui->zippedBytes->setText(QString::number(compr_data_size));
00073     if(inputFileName.isEmpty()) {
00074         ui->percentage->setText("");
00075         return;
00076     }
00077     double perc = (compr_data_size + 14 + val) * 100 / (size * 3) *
      bitsUsed / 8;
00078     ui->percentage->setText(QString::number(perc) + "%");
00079     goodPercentage = perc < 70;
00080 }
00085 void EncryptDialog::on_buttonBox_accepted()
00086 {
00087     if(!goodPercentage) {
00088         alert("Your encoding percentage is over 70% which is a bit ambiguous :(");
00089         success = false;
00090         return;
00091     }
00092     // Final zip
00093     compr_data = zip();
00094     success = true;
00095     close();
00096 }
00100 void EncryptDialog::on_buttonBox_rejected()
00101 {
00102     success = false;
00103     close();
00104 }
00110 void EncryptDialog::on_horizontalSlider_valueChanged(int
      value)
00111 {
00112     // Key generator with value of charachters
00113     key.clear();
00114     for(int i = 0; i < value; i++)
00115         key.append(48 + qrand() % 75);
00116     val = value;
00117     ui->keyLabel->setText(QString::number(value));
00118 }
00123 void EncryptDialog::on_bitsSlider_valueChanged(int value)
00124 {
00125     bitsUsed = value;
00126     ui->bitsUsedLbl->setText(QString::number(value));
00127     if(ui->percentage->text().isEmpty())
00128         return;
00129     double perc = (compr_data.size() + 14 + val) * 100 / (size * 3) * 8 /
      bitsUsed;
00130     ui->percentage->setText(QString::number(perc) + "%");
00131 }
```

9.15 C:/Users/salex/Documents/GitHub/PictureCrypt/encryptdialog.h File Reference

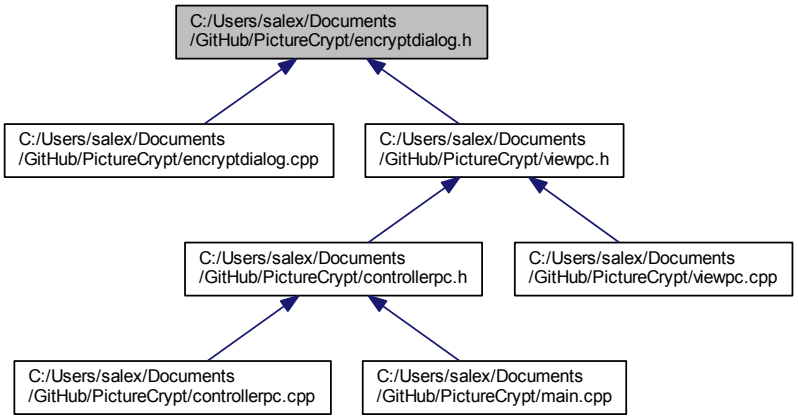
```
#include <QDialog>
```

```
#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

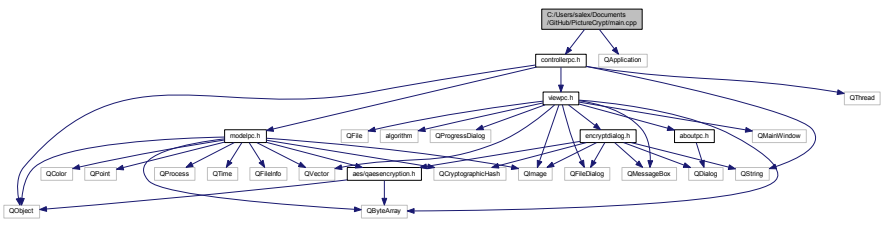
9.16 encryptdialog.h

```
00001 #ifndef ENCRYPTDIALOG_H
00002 #define ENCRYPTDIALOG_H
00003
00004 #include <QDialog>
00005 #include <QFileDialog>
00006 #include <QImage>
00007 #include <QMessageBox>
00008 #include <QString>
00009
00010 #include <aes/qaesencryption.h>
00011 #include <QCryptographicHash>
00012
00013 namespace Ui {
00014 class EncryptDialog;
00015 }
00021 class EncryptDialog : public QDialog
00022 {
00023     Q_OBJECT
00024
00025 public:
00026     explicit EncryptDialog(QByteArray _data, QWidget *parent = 0);
00027     ~EncryptDialog();
00028
00029 public slots:
00030     void on_fileButton_clicked();
00031
00032     void on_buttonBox_accepted();
00033
00034     void on_buttonBox_rejected();
00035
00036     void on_horizontalSlider_valueChanged(int value);
00037
00038 public:
00042     QByteArray data;
00046     bool success;
00050     QByteArray compr_data;
00054     QString inputFileName;
00058     long long int size;
00062     QString key;
00066     bool goodPercentage;
00070     int val;
00075     int bitsUsed;
00079     QImage image;
00080     QByteArray zip();
00081 private slots:
00082     void on_bitsSlider_valueChanged(int value);
00083
00084 private:
00085     Ui::EncryptDialog *ui;
00086     void alert(QString text);
00087 };
00088
00089 #endif // ENCRYPTDIALOG_H
```

9.17 C:/Users/salex/Documents/GitHub/PictureCrypt/main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

9.17.1 Function Documentation

9.17.1.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 111 of file main.cpp.

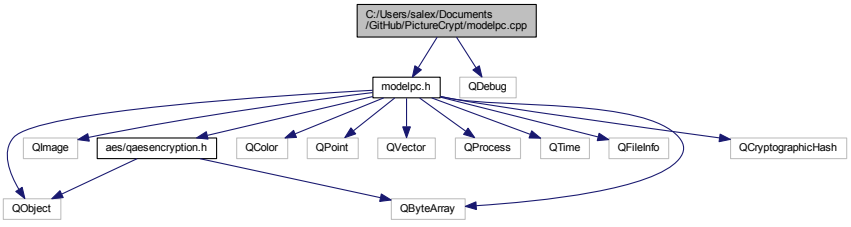
9.18 main.cpp

```
00001 #include "controllerpc.h"
00002 #include <QApplication>
00111 int main(int argc, char *argv[])
00112 {
00113     QApplication a(argc, argv);
00114     ControllerPC w;
00115
00116     return a.exec();
00117 }
```

9.19 C:/Users/salex/Documents/GitHub/PictureCrypt/modelpc.cpp File Reference

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

Include dependency graph for modelpc.cpp:



9.20 modelpc.cpp

```
00001 #include "modelpc.h"
00002 #include <QDebug>
00008 ModelPC::ModelPC(long _version)
00009 {
00010     // Version control
00011     version = _version;
00012     ver_byte = bytes((long int) (version / 65536)) + bytes((long int) (
      version / 256) % 256);
00013     ver_byte += bytes(version % 256);
00014     qsrand(randSeed());
00015 }
00026 void ModelPC::start(QByteArray data, QImage image, int _bitsUsed, QString key, int mode)
00027 {
00028     if(key.isEmpty()) {
00029         qsrand(randSeed());
00030         for(int i = 0; i < 64; i++)
00031             key.append(48 + qrand() % 75);
00032     }
00033     else if(key.size() > 255) {
00034         alert("Key is too big, max is 255 bytes", true);
00035         return;
00036     }
00037     long long usedBytes = data.size() + 14 + key.size();
00038     long long size = image.width() * image.height();
00039     if(usedBytes * 100 / (size * 3) * 8 / _bitsUsed > 70) {
00040         alert("Too much data for this image", true);
00041         return;
00042     }
00043
00044     curMode = mode;
00045     bitsUsed = _bitsUsed;
00046
00047     QByteArray key_data = key.toUtf8();
00048     QByteArray zipped_data = zip(data, key_data);
00049     QByteArray encr_data = bytes(key_data.size()) + key_data + zipped_data;
00050
00051     encrypt(encr_data, &image);
00052 }
00053
00061 void ModelPC::encrypt(QByteArray encr_data, QImage * image, int mode)
00062 {
00063     // TODO Remove debug mode = 0
00064     mode = 0;
00065
00066     encr_data = ver_byte + encr_data;
00067     long long int countBytes = encr_data.size();
00068     curMode = mode;
00069     switch(curMode)
00070     {
00071     case 0:
00072         circuit(image, &encr_data, countBytes);
00073         break;
00074     case 1:
00075         jphs(image, &encr_data);
00076         break;
00077     default:
00078         fail("Something went wrong! Error code 4.");
00079         return;
00080     }
00081
00082
00083     // Saving
00084     if(success)
00085         emit saveImage(image);
00086 }
00092 void ModelPC::decrypt(QImage * image)
00093 {
00094     // Image opening
00095     int w = image->width();
00096     int h = image->height();
00097
00098     // Getting corner pixels
00099     QColor colUL = image->pixelColor(0, 0).toRgb();
00100     QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00101     QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00102
00103     // Getting verification code
00104     int verifCode = (((colUR.green() % 2) << 5) + colUR.blue() % 32) << 2;
00105     verifCode += colDR.blue() % 4;
00106     if(verifCode != 166){
00107         alert("Image wasn't encrypted by this app or is damaged!");
00108         return;
00109     }
00110     // Getting number of bytes
```

```
00111     long long int countBytes = (colUL.blue() % 32 + ((colUL.green() % 32) << 5) + ((colUL.red() % 32) << 10
      )) << 9;
00112     countBytes += ((colUR.red() % 32) << 4) + (colUR.green() >> 1) % 16;
00113
00114     bitsUsed = (colDR.blue() >> 2) % 8 + 1;
00115     curMode = colDR.green() % 32;
00116
00117     // Start of the circuit
00118     QByteArray data;
00119     circuit(image, &data, countBytes);
00120
00121     // Check if circuit was successful
00122     if(!success)
00123         return;
00124     if(data.isEmpty())
00125     {
00126         alert("Read data is empty. Cannot continue!", true);
00127         return;
00128
00129     }
00130     // Version check
00131     long long int _ver = mod(data.at(0) * pow(2, 16));
00132     _ver += mod(data.at(1) * pow(2, 8));
00133     _ver += mod(data.at(2));
00134     data.remove(0, 3);
00135     if(_ver > version) {
00136         alert("Picture's app version is newer than yours. Image version is "
00137               + generateVersionString(_ver) + ", yours is "
00138               + generateVersionString(version) + ".", true);
00139         return;
00140     }
00141     else if(_ver < version) {
00142         alert("Picture's app version is older than yours. Image version is "
00143               + generateVersionString(_ver) + ", yours is "
00144               + generateVersionString(version) + ".", true);
00145         return;
00146     }
00147
00148     // Obtain the key
00149     int key_size = mod(data.at(0));
00150     QByteArray key = data.mid(1, key_size);
00151     data.remove(0, key_size + 1);
00152
00153     // Unzip
00154     QByteArray unzipped_data = unzip(data, key);
00155     emit saveData(unzipped_data);
00156 }
00161 void ModelPC::fail(QString message)
00162 {
00163     alert(message, true);
00164     success = false;
00165     emit setProgress(101);
00166     return;
00167 }
00173 void ModelPC::jphs(QImage *image, QByteArray *data)
00174 {
00175     // Under Development
00176     return;
00177
00178     // Dead code
00179
00180     success = true;
00181     bool isEncrypt = !data->isEmpty();
00182     QString targetEXE = defaultJPHSDir + (isEncrypt ? "/jphide.exe" : "/jpseek.exe");
00183     if(!fileExists(targetEXE))
00184     {
00185         alert("JPHS not installed, installation required!\nSee Menu -> Configure -> JPHS directory",
      true);
00186         return;
00187     }
00188
00189     QString randomFileName = defaultJPHSDir + "/";
00190     qsrand(randSeed());
00191     for(int i = 0; i < 10; i++)
00192         randomFileName.append(97 + qrand() % 25);
00193     image->save(randomFileName + ".jpg");
00194     if(isEncrypt) {
00195         QFile file(randomFileName + ".pc");
00196         if(!file.open(QFile::WriteOnly)) {
00197             alert("Cannot save file, wait wut?",true);
00198             return;
00199         }
00200         file.write(*data);
00201         file.close();
00202
00203         QStringList args;
00204         args << (randomFileName + ".jpg") << (randomFileName + "_out.jpg") << (randomFileName + ".pc");
```

```
00205         QProcess prog(this);
00206         prog.start(targetEXE, args);
00207         prog.waitForStarted();
00208         prog.write("test\n");
00209         prog.waitForBytesWritten();
00210         prog.write("test\n");
00211         prog.waitForBytesWritten();
00212         prog.waitForReadyRead();
00213         QByteArray bytes = prog.readAll();
00214         prog.waitForFinished();
00215         //QByteArray readData = prog.readAll();
00216         prog.close();
00217         // Cleaning - Deleting temp files
00218
00219     }
00220     else {
00221
00222     }
00223
00224 }
00225
00234 void ModelPC::circuit(QImage *image, QByteArray *data, long long countBytes)
00235 {
00236
00237     // Some flags and creation of the ProgressDialog
00238     success = true;
00239     emit setProgress(-1);
00240     bool isEncrypt = !data->isEmpty();
00241
00242     // Image setup
00243     int w = image->width();
00244     int h = image->height();
00245
00246     // Visited pixels array
00247     QVector <QPoint> were;
00248     were.push_back(QPoint(0, 0));
00249     were.push_back(QPoint(0, h - 1));
00250     were.push_back(QPoint(w - 1, 0));
00251     were.push_back(QPoint(w - 1, h - 1));
00252
00253     long long int offset = 0;
00254
00255     // Pre-start Cleaning
00256     circuitData = data;
00257     circuitImage = image;
00258     circuitCountBytes = countBytes;
00259     cur = 0;
00260     bitsBuffer.clear();
00261
00262     // Writing Top-Left to Bottom-Left
00263     for(int i = 1; i < h - 1 && mustGoOn(isEncrypt); i++) {
00264         QPoint pos(0, i);
00265         processPixel(pos, &were, isEncrypt);
00266     }
00267     // Writing Bottom-Right to Top-Right
00268     if(mustGoOn(isEncrypt))
00269     {
00270         for(int i = h - 2; i >= 1 && mustGoOn(isEncrypt); i--){
00271             QPoint pos(w - 1, i);
00272             processPixel(pos, &were, isEncrypt);
00273         }
00274     }
00275     // Main cycle
00276     // Strong is considered as actual corner pixel and weak as pixel near it like (1, 0) or (0, 1)
00277     while(mustGoOn(isEncrypt))
00278     {
00279         // Strong Top-Right to Strong Bottom-Right
00280         for(int i = offset; i < h - offset && mustGoOn(isEncrypt); i++){
00281             QPoint pos(w - offset - 2, i);
00282             processPixel(pos, &were, isEncrypt);
00283         }
00284         // Strong Top-Left to Weak Top-Right
00285         for(int i = offset + 1; i < w - offset - 2 && mustGoOn(isEncrypt); i++){
00286             QPoint pos(i, offset);
00287             processPixel(pos, &were, isEncrypt);
00288         }
00289         // Weak Bottom-Right to Weak Bottom-Left
00290         for(int i = w - 3 - offset; i >= offset + 2 && mustGoOn(isEncrypt); i--){
00291             QPoint pos(i, h - offset - 1);
00292             processPixel(pos, &were, isEncrypt);
00293         }
00294         // Weak Top-Left to Strong Bottom-Left
00295         for(int i = offset + 1; i < h - offset && mustGoOn(isEncrypt); i++){
00296             QPoint pos(offset + 1, i);
00297             processPixel(pos, &were, isEncrypt);
00298         }
00299         offset++;
```

```
00300     }
00301     // Extra writing
00302     if(!success)
00303         return;
00304     if(isEncrypt)
00305     {
00306         // Getting past colors
00307         QColor colUL = image->pixelColor(0, 0).toRgb();
00308         QColor colUR = image->pixelColor(w - 1, 0).toRgb();
00309         QColor colDL = image->pixelColor(0, h - 1).toRgb();
00310         QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00311         int red = 0;
00312         int green = 0;
00313         int blue = 0;
00314
00315         // Writing Upper Left
00316         red = (colUL.red() & 224) + (countBytes >> 19);
00317         green = (colUL.green() & 224) + (countBytes >> 14) % 32;
00318         blue = (colUL.blue() & 224) + (countBytes >> 9) % 32;
00319         image->setPixelColor(0, 0, QColor(red, green, blue));
00320
00321         // Writing Upper Right
00322         red = (colUR.red() & 224) + (countBytes >> 4) % 32;
00323         green = (colUR.green() & 224) + ((countBytes % 16) << 1) + 1;
00324         blue = (colUR.blue() & 224) + 9;
00325         image->setPixelColor(w - 1, 0, QColor(red, green, blue));
00326
00327         // Getting extra bytes if left
00328         while(cur < countBytes)
00329             push(mod(circuitData->at(cur++)), 8);
00330         if(bitsBuffer.size() > 20) {
00331             fail("Something went wrong! Error code 1");
00332             return;
00333         }
00334         // Getting extra data as long.
00335         long extraData = pop(-2);
00336
00337         // Writing Down Left
00338         red = (colDL.red() & 224) + (extraData >> 15);
00339         green = (colDL.green() & 224) + (extraData >> 10) % 32;
00340         blue = (colDL.blue() & 224) + (extraData >> 5) % 32;
00341         image->setPixelColor(0, h - 1, QColor(red, green, blue));
00342
00343         // Writing Down Right
00344         red = (colDR.red() & 224) + extraData % 32;
00345         green = (colDR.green() & 224);
00346         blue = (colDR.blue() & 224) + ((bitsUsed - 1) << 2) + 2;
00347         image->setPixelColor(w - 1, h - 1, QColor(red, green, blue));
00348     }
00349     else
00350     {
00351         // Read the past pixels
00352         QColor colDL = image->pixelColor(0, h - 1).toRgb();
00353         QColor colDR = image->pixelColor(w - 1, h - 1).toRgb();
00354
00355         // Read extra data
00356         long extraData = ((colDL.red() % 32) << 15) + ((colDL.green() % 32) << 10);
00357         extraData += ((colDL.blue() % 32) << 5) + colDR.red() % 32;
00358
00359         // Add extra data to the bitsBuffer
00360         push(extraData, (countBytes - cur) * 8 - bitsBuffer.size());
00361
00362         // Move bits from bitsBuffer to the QByteArray
00363         while(!bitsBuffer.isEmpty())
00364             data->append(pop(8));
00365     }
00366     emit setProgress(101);
00367 }
00368
00376 void ModelPC::processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt)
00377 {
00378     if(!success)
00379         return;
00380     // Check if point was already visited
00381     if(were->contains(pos)){
00382         fail("Point (" + QString::number(pos.x()) + "," + QString::number(pos.y()) + ") was visited
      twice! Error code 2");
00383         return;
00384     }
00385     else
00386         were->push_back(pos);
00387     if(isEncrypt)
00388     {
00389         // Make sure that there are enough bits in bitsBuffer to write
00390         while(bitsBuffer.size() < 3 * bitsUsed)
00391             push(mod(circuitData->at(cur++)), 8);
00392         // Read past contains
```

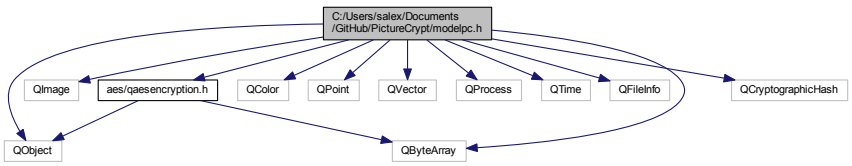
```
00393         QColor pixelColor = circuitImage->pixelColor(pos);
00394         int red = pixelColor.red();
00395         int green = pixelColor.green();
00396         int blue = pixelColor.blue();
00397 
00398         // Write new data in last bitsUsed pixels
00399         red += pop() - red % (int) pow(2, bitsUsed);
00400         green += pop() - green % (int) pow(2, bitsUsed);
00401         blue += pop() - blue % (int) pow(2, bitsUsed);
00402 
00403         circuitImage->setPixelColor(pos, QColor(red, green, blue));
00404     }
00405     else
00406     {
00407         QColor read_color = circuitImage->pixelColor(pos).toRgb();
00408         // Reading the pixel
00409         int red = read_color.red();
00410         int green = read_color.green();
00411         int blue = read_color.blue();
00412 
00413         // Reading the last bitsUsed pixels
00414         red %= (int) pow(2, bitsUsed);
00415         green %= (int) pow(2, bitsUsed);
00416         blue %= (int) pow(2, bitsUsed);
00417 
00418         // Getting the data in the bitsBuffer.
00419         push(red);
00420         push(green);
00421         push(blue);
00422 
00423         // Getting data to QByteArray
00424         while(bitsBuffer.size() >= 8) {
00425             circuitData->append(pop(8));
00426             cur++;
00427         }
00428     }
00429     emit setProgress(100 * cur / circuitCountBytes);
00430 }
00431 
00432 long ModelPC::pop(int bits)
00433 {
00434     // Hard to say
00435     long res = 0;
00436     int poppedBits = bits == -1 ? bitsUsed : bits;
00437     if(bits == -2)
00438         poppedBits = bitsBuffer.size();
00439     for(int i = 0; i < poppedBits; i++)
00440         res += bitsBuffer[i] * pow(2, poppedBits - i - 1);
00441     bitsBuffer.remove(0, poppedBits);
00442     return res;
00443 }
00444 
00445 void ModelPC::push(int data, int bits)
00446 {
00447     // That's easier, but also hard
00448     int buf_size = bitsBuffer.size();
00449     int extraSize = bits == -1 ? bitsUsed : bits;
00450     bitsBuffer.resize(buf_size + extraSize);
00451     for(int i = bitsBuffer.size() - 1; i >= buf_size; i--, data >>= 1)
00452         bitsBuffer[i] = data % 2;
00453 }
00454 
00455 bool ModelPC::mustGoOn(bool isEncrypt)
00456 {
00457     return success && (isEncrypt ? (circuitCountBytes - cur) * 8 + bitsBuffer.size() >=
    bitsUsed * 3 :
00458                                    circuitData->size() * 8 + bitsBuffer.size() <
00459                                    circuitCountBytes * 8 - (circuitCountBytes * 8)% (
    bitsUsed * 3));
00460 }
00469 QByteArray ModelPC::unzip(QByteArray data, QByteArray key)
00470 {
00471     // Decryption
00472     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00473     QAESEncryption encryption(QAESEncryption::AES_256,
    QAESEncryption::ECB);
00474     QByteArray new_data = encryption.decode(data, hashKey);
00475     // Decompressing
00476     return qUncompress(new_data);
00477 }
00486 QByteArray ModelPC::zip(QByteArray data, QByteArray key)
00487 {
00488     // Zip
00489     QByteArray c_data = qCompress(data, 9);
00490     // Encryption
00491     QByteArray hashKey = QCryptographicHash::hash(key, QCryptographicHash::Sha256);
00492     return QAESEncryption::Crypt(QAESEncryption::AES_256,
```

```
    QAESEncryption::ECB, c_data, hashKey);
00493 }
00494
00495 bool ModelPC::fileExists(QString path)
00496 {
00497     QFileInfo check_file(path);
00498     return check_file.exists() && check_file.isFile();
00499 }
00500
00507 QByteArray ModelPC::bytes(long long n)
00508 {
00509     return QByteArray::fromHex(QByteArray::number(n, 16));
00510 }
00517 unsigned int ModelPC::mod(int input)
00518 {
00519     if(input < 0)
00520         return (unsigned int) (256 + input);
00521     else
00522         return (unsigned int) input;
00523 }
00530 void ModelPC::alert(QString message, bool isWarning)
00531 {
00532     emit alertView(message, isWarning);
00533 }
00539 QColor ModelPC::RGBbytes(long long byte)
00540 {
00541     int blue = byte % 256;
00542     int green = (byte / 256) % 256;
00543     int red = byte / pow(2, 16);
00544     return QColor(red, green, blue);
00545 }
00546
00547 QString ModelPC::generateVersionString(long ver)
00548 {
00549     return QString::number((int)( ver / pow(2, 16))) + "." + QString::number(((int) (ver / 256)) % 256) + "
    ." + QString::number(ver % 256);
00550 }
00551
00552 uint ModelPC::randSeed()
00553 {
00554     QTime time = QTime::currentTime();
00555     uint randSeed = time.msecsSinceStartOfDay() % 65536 + time.minute() * 21 + time.second() * 2;
00556     return randSeed;
00557 }
```

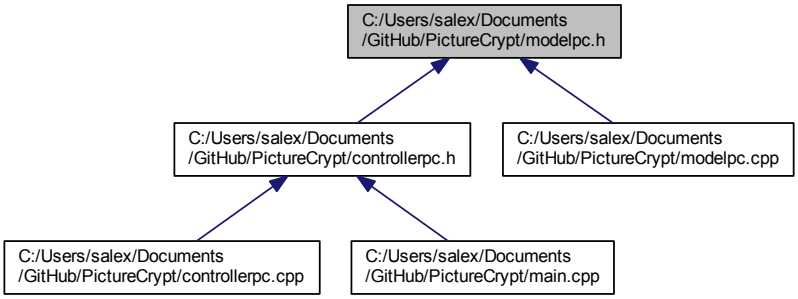
9.21 C:/Users/salex/Documents/GitHub/PictureCrypt/modelpc.h File Reference

```
#include <QObject>
#include <QImage>
#include <QByteArray>
#include <QColor>
#include <QPoint>
#include <QVector>
#include <QProcess>
#include <QTime>
#include <QFileInfo>
#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. Controled by ControllerPC.

9.21.1 Detailed Description

Header of ModelPC class

See also

ControllerPC, ModelPC, ViewPC

Definition in file modelpc.h.

9.22 modelpc.h

```
00001 #ifndef MODELPC_H
00002 #define MODELPC_H
00003
00004 #include <QObject>
00005 #include <QImage>
00006 #include <QByteArray>
00007 #include <QColor>
00008 #include <QPoint>
00009 #include <QVector>
00010 #include <QProcess>
00011 #include <QTime>
00012 #include <QFileInfo>
00013
00014 #include <aes/qaesencryption.h>
00015 #include <QCryptographicHash>
00026 class ModelPC : public QObject
00027 {
00028     Q_OBJECT
00029 public:
00030     ModelPC(long _version = 131072);
00031
00032 signals:
00039     alertView(QString message, bool isWarning);
00044     saveData(QByteArray data);
00049     saveImage(QImage *image);
00055     setProgress(int val);
```

```
00056 public slots:
00057     void start(QByteArray data, QImage image, int mode = 0, QString key = "", int _bitsUsed = 8);
00058     void encrypt(QByteArray encr_data, QImage * image, int mode = 0);
00059     void decrypt(QImage * image);
00060     void fail(QString message);
00061
00062 public:
00067     bool success;
00071     long version;
00075     int curMode;
00079     int bitsUsed;
00083     QString defaultJPHSDir;
00084     QByteArray unzip(QByteArray data, QByteArray key);
00085     void alert(QString message, bool isWarning = false);
00086 protected:
00087     void circuit(QImage * image, QByteArray * data, long long int countBytes);
00088     void jphs(QImage * image, QByteArray * data);
00089     void processPixel(QPoint pos, QVector<QPoint> *were, bool isEncrypt);
00090     QByteArray zip(QByteArray data, QByteArray key);
00091 private:
00092     bool fileExists(QString path);
00093     QByteArray bytes(long long n);
00094     unsigned int mod(int input);
00095     QByteArray ver_byte;
00096     QColor RGBbytes(long long byte);
00097     QString generateVersionString(long ver);
00098     uint randSeed();
00099
00100     QByteArray * circuitData;
00101     QImage * circuitImage;
00102     long long circuitCountBytes;
00103     long cur;
00104     bool mustGoOn(bool isEncrypt);
00105
00106     QVector <bool> bitsBuffer;
00107     long pop(int bits = -1);
00108     void push(int data, int bits = -1);
00109
00110 };
00111
00112 #endif // MODELPC_H
```

9.23 C:/Users/salex/Documents/GitHub/PictureCrypt/README.md File Reference

9.24 C:/Users/salex/Documents/GitHub/PictureCrypt/README.md

```
00001 # PictureCrypt
00002 Make your pictures crypted.
00003
00004 ## External usage
00005 ModelPC class can be used externally (without UI)
00006 ```
00007 #include <modelpc.h>
00008 #include <QByteArray>
00009 #include <QImage>
00010
00011 ...
00012
00013 ModelPC * model = new ModelPC(ver);
00014 // ver is version of the app, used to check the data structure version
00015 // ver is type long and is calculated as if version is "x.y.z" => ver = x * 65536 + y * 256 + z
00016 // Default parameter is 2^17 (2.0.0)
00017
00018 // Connecting signals
00019
00020 // Essential ones
00021
00022 model->saveData(QByteArray data)
00023 // Used to return the retrieved data
00024
00025 model->saveImage(QImage * image)
00026 // Used to return the modified image
00027
00028 // Extra ones
00029
00030 model->alertView(QString message, bool isWarning)
00031 // Used for messages to be shown to users
00032
00033 model->setProgress(int val)
```

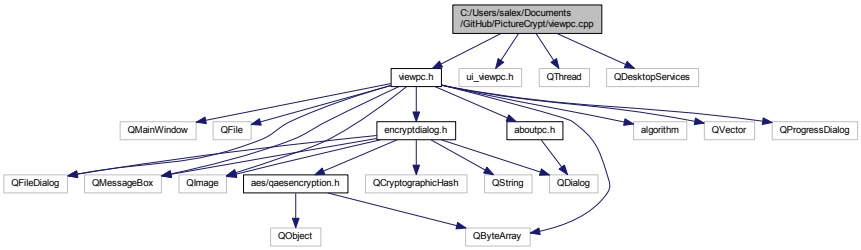
```
00034 // Used to show user the progress of embedding
00035 // -1 indicates the creation of some kind of progress dialog
00036 // from 0 to 100 shows the progress
00037 // 101 indicates that progress dialog should be closed
00038
00039 ```
00040
00041 ## Avaible methods
00042 ### Essential ones
00043 #### start
00044 Used for embedding
00045
00046 Parameters:
00047 data   Data to be encrypted
00048 _image Image to be encrypted into.
00049 _bitsUsed  Bits per byte, see also ModelPC::bitsUsed
00050 key    Key, if default (empty), random key of 64 charachters will be generated.
00051 mode   Mode of encryption
00052 ```
00053 model->start(QByteArray data, QImage image, int mode = 0, QString key = "", int _bitsUsed = 8);
00054 ```
00055
00056 #### decrypt
00057 Used for de-embedding
00058
00059 Parameters:
00060 image  Image to be decrypted.
00061
00062 ```
00063 model->decrypt(QImage * image);
00064 ```
00065 ### Extra ones
00066 #### encrypt
00067 Used for embedding but with data already packed with stuff like version, file size, aes key, etc.
00068 Used in PictureCrypt project
00069
00070 Parameters:
00071
00072 encr_data  Data to be embbed to an image.
00073 image  Image to be embbed into.
00074 mode   Mode of encryption
00075
00076 ```
00077 model->encrypt(QByteArray encr_data, QImage * image, int mode = 0);
00078 ```
00079 #### fail
00080 Used for stopping the embedding or de-embedding proccess
00081 Parameters:
00082
00083 message    Message for user
00084 ```
00085 model->fail(QString message);
00086 ```
00087
00088 ## Avaible modes of embedding
00089 * 0 - Standard, created by me
00090 * 1 - JPHS, requires manually installed JPHS and specified directory.
00091
00092 ## Documentation
00093 Doxygen Documentation avaibale [here](http://doc.alex.unaux.com/picturecrypt)
00094
00095 ## Windows Installer
00096 Windows installer for non-QT build [here](http://bit.ly/PictureCryptProject)
00097
00098 ## Dependancies
00099 * qtcore
00100 * [QAESEncryption](https://github.com/bricke/Qt-AES) by bricke
00101
00102 ## Contact
00103 Question or suggestions are welcome!
00104 Please use the GitHub issue tracking to report suggestions or issues.
00105 Email me a.kovrigin0@gmail.com and visit my site http://alex.unaux.com
00106
00107 ## License
00108 This software is provided under the [UNLICENSE](http://unlicense.org/)
```

9.25 C:/Users/salex/Documents/GitHub/PictureCrypt/viewpc.cpp File Reference

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

```
#include <QThread>
#include <QDesktopServices>
```

Include dependency graph for viewpc.cpp:



9.26 viewpc.cpp

```
00001 #include "viewpc.h"
00002 #include "ui_viewpc.h"
00003 #include <QThread>
00004 #include <QDesktopServices>
00005
00006 ViewPC::ViewPC(QWidget *parent) :
00007     QMainWindow(parent),
00008     ui(new Ui::ViewPC)
00009 {
00010     ui->setupUi(this);
00011 }
00012
00013 ViewPC::~ViewPC()
00014 {
00015     delete ui;
00016 }
00017
00018 void ViewPC::on_encryptMode_clicked()
00019 {
00020     // Encrypt radio button clicked
00021     setEncryptMode(true);
00022 }
00023
00024 void ViewPC::on_decryptMode_clicked()
00025 {
00026     // Decrypt radio button clicked
00027     setEncryptMode(false);
00028 }
00032 void ViewPC::on_fileButton_clicked()
00033 {
00034     // Opening QFileDialog depending on isEncrypt
00035     if(isEncrypt)
00036         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.txt", tr("Text
      files (*.txt);;All Files (*)"));
00037     else
00038         inputFileName = QFileDialog::getOpenFileName(this, tr("Select file"), "/untitled.png", tr("PNG
      files (*.png);;All Files (*)"));
00039     // Display the file name
00040     ui->fileLabel->setText(inputFileName.isEmpty() ? "File not chosen" : inputFileName);
00041 }
00054 void ViewPC::on_startButton_clicked()
00055 {
00056     if(isEncrypt)
00057     {
00058         // Getting the data
00059         QString text = ui->text->toPlainText();
00060         QByteArray data;
00061         if(text.isEmpty()) {
00062             if(inputFileName.isEmpty()) {
00063                 alert("No input file or text was not given. Cannot continue!", true);
00064                 return;
00065             }
00066             // Opening the file
00067             QFile file(inputFileName);
00068             if (!file.open(QIODevice::ReadOnly))
00069             {
00070                 alert("Cannot open file. Cannot continue!", true);
```

```
00071                 return;
00072             }
00073             // Check the data size
00074             auto size = file.size();
00075             if(size > pow(2, 24)) {
00076                 alert("Your file is too big, our systems can handle it, but it requires a lot of time.
      We decline.", true);
00077                 file.close();
00078                 return;
00079             }
00080             data = file.readAll();
00081             file.close();
00082         }
00083         else
00084             data = text.toUtf8();
00085         // Select image via EncryptDialog
00086         EncryptDialog * dialog = new EncryptDialog(data);
00087         dialog->exec();
00088         if(!dialog->success)
00089             return;
00090 
00091         // Get the data
00092         QByteArray encr_data = dialog->compr_data;
00093 
00094         // Save the key
00095         QByteArray key_data = dialog->key.toUtf8();
00096 
00097         encr_data = bytes(key_data.size()) + key_data + encr_data;
00098         // TODO do the mode thing
00099         emit setBitsUsed(dialog->bitsUsed);
00100         emit encrypt(encr_data, &dialog->image, 0);
00101     }
00102     else
00103     {
00104         // Get the filename of the image
00105         if(!ui->text->toPlainText().isEmpty())
00106             alert("Obviously, the text browser isn't supported for decryption, use File Dialog
      instead.");
00107         if(inputFileName.isEmpty()) {
00108             alert("File not selected. Cannot continue!", true);
00109             return;
00110         }
00111         QImage * res_image = new QImage(inputFileName);
00112         emit decrypt(res_image);
00113     }
00114 }
00120 void ViewPC::alert(QString message, bool isWarning)
00121 {
00122     // Create message box
00123     QMessageBox box;
00124     if(isWarning)
00125         box.setIcon(QMessageBox::Warning);
00126     else
00127         box.setIcon(QMessageBox::Information);
00128     box.setText(message);
00129     box.setWindowIcon(QIcon(":/icons/mail.png"));
00130     box.setWindowTitle("Message");
00131     box.exec();
00132 }
00138 void ViewPC::saveData(QByteArray Edata)
00139 {
00140     // Save data using QFileDialog
00141     QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save File"),
00142                                 "/untitled.txt",
00143                                 tr("Text(*.txt);;All files (*)"));
00144     QFile writeFile(outputFileName);
00145     if (!writeFile.open(QIODevice::WriteOnly))
00146     {
00147         alert("Cannot access file path. Cannot continue!", true);
00148         return;
00149     }
00150     writeFile.write(Edata);
00151     writeFile.close();
00152     alert("Decryption completed!");
00153 }
00159 void ViewPC::saveImage(QImage * image)
00160 {
00161     // Save image using QFileDialog
00162     QString outputFileName = QFileDialog::getSaveFileName(this, tr("Save Image"),
00163                                 "/untitled.png",
00164                                 tr("Images(*.png)"));
00165     if(!image->save(outputFileName)) {
00166         alert("Cannot save file. Unable to continue!", true);
00167         return;
00168     }
00169     alert("Encryption completed!");
00170 }
```

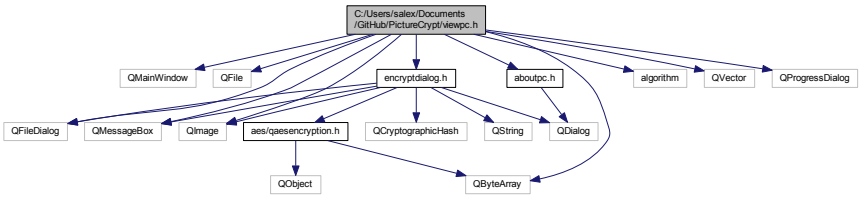
```
00177 void ViewPC::setProgress(int val)
00178 {
00179     if(val < 0) {
00180         // Create dialog
00181         dialog = new QProgressDialog("Cryption in progress.", "Cancel", 0, 100);
00182         connect(dialog, SIGNAL(canceled()), this, SLOT(abortCircuit()));
00183         progressDialogClosed = false;
00184         dialog->setWindowTitle("Processing");
00185         dialog->setWindowIcon(QIcon(":/icons/loading.png"));
00186         dialog->show();
00187     }
00188     else if(val >= 100) {
00189         // Close dialog
00190         dialog->setValue(100);
00191         QThread::msleep(25);
00192         dialog->close();
00193         dialog->reset();
00194         progressDialogClosed = true;
00195     }
00196     // Update the progress
00197     else if(!progressDialogClosed)
00198         dialog->setValue(val);
00199 }
00203 void ViewPC::abortCircuit()
00204 {
00205     // Set the flag
00206     progressDialogClosed = true;
00207     // Close the dialog
00208     dialog->close();
00209     dialog->reset();
00210     emit abortModel();
00211 }
00216 void ViewPC::setEncryptMode(bool encr)
00217 {
00218     ui->text->setEnabled(encr);
00219     isEncrypt = encr;
00220 }
00225 void ViewPC::setVersion(QString version)
00226 {
00227     // Version setup
00228     versionString = version;
00229 }
00230
00231 QByteArray ViewPC::bytes(long long n)
00232 {
00233     return QByteArray::fromHex(QByteArray::number(n, 16));
00234 }
00238 void ViewPC::on_actionAbout_triggered()
00239 {
00240     AboutPC about;
00241     about.setVersion(versionString);
00242     about.exec();
00243 }
00244
00248 void ViewPC::on_actionHelp_triggered()
00249 {
00250     QUrl docLink("http://doc.alex.unaux.com/picturecrypt");
00251     QDesktopServices::openUrl(docLink);
00252 }
00253
00254 void ViewPC::on_actionJPHS_path_triggered()
00255 {
00256     QString dir = QFileDialog::getExistingDirectory(this, tr("Open JPHS folder"),
00257                                                     "/home",
00258                                                     QFileDialog::ShowDirsOnly
00259                                                     | QFileDialog::DontResolveSymlinks);
00260     emit setJPHSDir(dir);
00261 }
```

9.27 C:/Users/salex/Documents/GitHub/PictureCrypt/viewpc.h File Reference

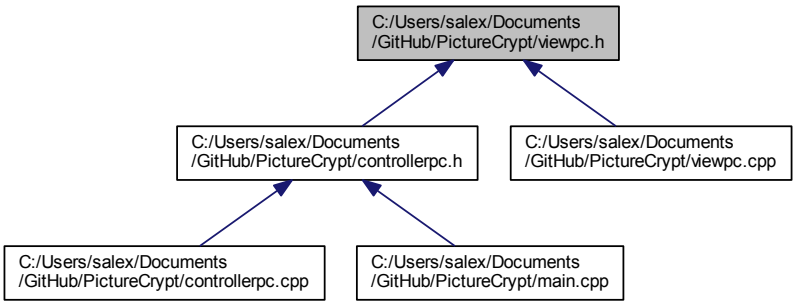
```
#include <QMainWindow>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QImage>
#include <algorithm>
#include <QByteArray>
```

```
#include <QVector>
#include <encryptdialog.h>
#include <QProgressDialog>
#include <aboutpc.h>
```

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

9.27.1 Detailed Description

Header of ViewPC class

See also

ControllerPC, ModelPC, ViewPC

Definition in file viewpc.h.

9.28 viewpc.h

```
00001 #ifndef VIEWPC_H
00002 #define VIEWPC_H
00003
00004 #include <QMainWindow>
00005 #include <QFile>
00006 #include <QFileDialog>
00007 #include <QMessageBox>
00008 #include <QImage>
00009 #include <algorithm>
00010 #include <QByteArray>
00011 #include <QVector>
00012
00013 #include <encryptdialog.h>
00014 #include <QProgressDialog>
00015 #include <aboutpc.h>
00016
00017 namespace Ui {
00018 class ViewPC;
00019 }
00029 class ViewPC : public QMainWindow
00030 {
00031     Q_OBJECT
00032
00033 public:
00034     explicit ViewPC(QWidget *parent = 0);
00035     ~ViewPC();
00036 private slots:
00037     void on_encryptMode_clicked();
00038
00039     void on_decryptMode_clicked();
00040
00041     void on_actionJPHS_path_triggered();
00042
00043 protected slots:
00044     void on_fileButton_clicked();
00045
00046     void on_startButton_clicked();
00047
00048     void on_actionAbout_triggered();
00049
00050     void on_actionHelp_triggered();
00051 public slots:
00052     void alert(QString message, bool isWarning = false);
00053     void saveData(QByteArray Edata);
00054     void saveImage(QImage *image);
00055     void setProgress(int val);
00056     void abortCircuit();
00057     void setEncryptMode(bool encr);
00058     void setVersion(QString version);
00059 signals:
00066     encrypt(QByteArray data, QImage * image, int mode);
00071     decrypt(QImage * _image);
00075     abortModel();
00081     setBitsUsed(int bitsUsed);
00086     setJPHSDir(QString dir);
00087 public:
00092     QProgressDialog * dialog;
00097     bool progressDialogClosed;
00098 private:
00099     Ui::ViewPC *ui;
00100     bool isEncrypt;
00101     QString inputFileName;
00102     QByteArray bytes(long long n);
00103     QString versionString;
00104 };
00105
00106 #endif // VIEWPC_H
```

Index