山东大学 软件 学院

**密码学原理与实践** 课程实验实践报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100150209 | 姓名：杨佳庆 | | 专业与班级：2021级网安班 |
| 实验实践编号： | | | |
| 实验实践题目：系列密码算法实现与应用集成 | | | |
| 实验实践学时：16 | | 实验日期：2023-2024-1学期 | |
| 实验实践目的：   1. 基于密码实验实践教学工具套件，结合软件工程，分析该系统存在的设计与实现问题，提出修改和改进完善的方案。 2. 通过一系列密码算法的独立编码接口封装实现（可借助于开源代码），熟悉并掌握常规的分组对称密码算法、序列对称密码算法、HASH函数、MAC函数、数字签名算法的接口编制方法，并能够利用可用编程接口针对文件加解密实现一个实用工具设计，要求体现：   （1）各种密码技术在方案设计中得到正确、合理、合适的集成应用；  （2）方案设计体现安全性与方便性的结合与平衡。  （3）可以针对任意类型、任意大小的文件操作。 | | | |
| 硬件环境：  设备名称 LAPTOP-UKNRNMI2  处理器 Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz  机带 RAM 16.0 GB (15.9 GB 可用)  设备 ID E6D4BD13-0B7C-4171-8511-F3F4F35DB823  产品 ID 00330-80128-19810-AA384  系统类型 64 位操作系统, 基于 x64 的处理器  笔和触控 笔支持 | | | |
| 软件环境：  操作系统：  版本 Windows 11 专业版  版本 23H2  操作系统版本 22635.2841  开发工具：  PyCharm 2023.2.1  使用的软件包以及版本：  详见软件环境要求.txt  *实验实践用到的操作系统、各种开发工具、开源软件的名称、版本。需要使读者能够按照该环境就可以还原实验实践软件环境，并能够验证实验实践结果的可重复性和正确性。* | | | |
| 实验实践步骤与内容：（包括或不限于）  *根据实验实践的内容，以及实现的过程，可写的内容包括：*  *实验实践的设计思路*  *实验实践模型的描述*  *实验实践功能说明*  *实验实践设计说明*  *主要数据结构的分析说明*  *主要算法代码的分析说明*  *项目管理文件的说明*  *实验实践投入的实际学时数*  *调试排错过程的记录*  *运行结果(文本格式，必要时抓屏)*  *多种方式测试结果的记录*  *实验结果的分析综合*  *……*  基于密码实验实践教学工具套件，结合软件工程，分析该系统存在的设计与实现问题，提出修改和改进完善的方案。  **概述**  本课程实验基于一套密码实验实践教学工具套件，再此基础上进行二次开发，实现需要的功能，同时对现有功能进行改进。  由于原本的代码是专用教学的，而且是公司产品，所以部分核心代码被封装到了软件公司提供的Key和智能卡片里了，如果不在其平台基于Key使用，直接将代码于Pycharm等开发工具则无法运行。  因此我的开发思路就是基于现有的代码，实现所缺少的那一部分核心代码，因为缺失的核心代码是利用Pyqt5实现的界面设计以及一些利用Key的加解密算法。  因此总的计划分三步走：  第一步，实现Pyqt5的界面设计代码。因为参考的代码每个都是基于相同的包实现的界面，因此只需要完成主窗口设计以及相应封装的Package就能够让大部分界面通过很少的改动就能够运行。  第二步，对现有缺失的算法进行重新编写，因为最终要实现的功能是不基于外部Key的，因此对其利用Key或者智能卡的加解密算法要摒弃掉，使用一些开放协议或者自己实现，例如Openssl等。  第三步，根据需求（针对系统缺陷，不符合密码学思想的设计），重新对界面和功能进行设计。同时增加一些特色内容，比如增加前端界面的样式；功能性增加对文件的加解密等。  **程序目录结构**  如下图所示：    更加详细的目录树可以查看dirtree.txt文件。  这里目录树可以通过CMD命令打印到文件，也可以写个Python程序打印。  **CMD生成目录树**  在 windows 系统中，有一个 CMD 指令可以生成目录树，该条指令是 " **tree** " 。  以图形显示驱动器或路径的文件夹结构。  TREE [drive:][path] [/F] [/A]  /F 显示每个文件夹中文件的名称。  /A 使用 ASCII 字符，而不使用扩展字符。  **Python实现：**  思路：给出一个路径，判断是否为文件，如果是，则字符串拼接；否则，字符串拼接并且递归查询该目录下的文件。字符串拼接的时候以缩进来显示层次结构。  tree\_str = ''    def generate\_tree(pathname, n=0):  global tree\_str  if pathname.is\_file():  tree\_str += ' |' \* n + '-' \* 4 + pathname.name + '\n'  elif pathname.is\_dir():  tree\_str += ' |' \* n + '-' \* 4 + \  str(pathname.relative\_to(pathname.parent)) + '\\' + '\n'  for cp in pathname.iterdir():  generate\_tree(cp, n + 1)    if \_\_name\_\_ == '\_\_main\_\_':  generate\_tree(Path.cwd())  print(tree\_str)  **代码实现：**  分析项目源码  from MathMagic.Modules.CryptographyModule import CryptographyWidget, Button, PlainTextEdit, IntroductionTab, IntermediateValueTab, SmartCardTab, SmartCard, Group, ErrorType  from Util import Path, TypeConvert  观察到每个ui文件都导入了MathMagic的CryptographyModule包，但下载的软件里并没有这个文件，询问得知这个包并不公开，属于开发公司。因此只好自己实现，而Util包和其他算法的包都是公开的，那么问题就很清楚了。就是自己手动实现被封装的包。  阅读大部分的源码后，发现其ui.py的实现大抵相同，只有具体的算法不同，ui.py主要负责窗口的绘制，还有一些设置类参数。随便找个ui文件进行说明。  AES\_ui.py  class UserDefinedWidget(CryptographyWidget):  def \_\_init\_\_(self, parent):  CryptographyWidget.\_\_init\_\_(self, parent)  self.setWindowTitle("AES")  self.tabs\_config = [IntroductionTab(  link="file:///" + Path.MENU\_DIRECTORY + "/CryptographicAlgorithm/BlockCipherAlgorithm/AES/html/index.html"),  IntermediateValueTab(), SmartCardTab()]  # set smart card widget configurations  self.smart\_card\_config = SmartCard()  # set groups configurations  # set plain text edit component configurations  # set button component configurations'  # id: the identity of the component  # clicked\_function: execute the function after the button clicked  self.groups\_config = [  Group(name="Key",  plain\_text\_edits=[Key(enabled=True, id="Key", label="Key (Hex)",  default\_text="2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C")],  buttons=[  Button(id="KeyGenerate", name="KeyGenerate", clicked\_function=self.generate\_key)  ]),  Group(name="Encrypt",  plain\_text\_edits=[PlainTextEdit(id="Plaintext", label="Plaintext (Hex)",  default\_text="32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34"),  PlainTextEdit(id="\_Ciphertext", label="Ciphertext (Hex)",  default\_text="", read\_only=True)],  buttons=[  Button(id="ComputerEncrypt", name="Encrypt (PC)", clicked\_function=self.computer\_encrypt),  Button(id="CardEncrypt", name="Encrypt (Card)", clicked\_function=self.card\_encrypt),  Button(id="CleanEncrypt", name="Clean", clicked\_function=self.encrypt\_clean)  ]),  Group(name="Decrypt",  plain\_text\_edits=[PlainTextEdit(id="Ciphertext", label="Ciphertext (Hex)",  default\_text=""),  PlainTextEdit(id="\_Plaintext", label="Plaintext (Hex)",  default\_text="", read\_only=True)],  buttons=[  Button(id="ComputerDecrypt", name="Decrypt (PC)", clicked\_function=self.computer\_decrypt),  Button(id="CardDecrypt", name="Decrypt (Card)", clicked\_function=self.card\_decrypt),  Button(id="CleanDecrypt", name="Clean", clicked\_function=self.decrypt\_clean)  ])  ]   # render user interface based on above-mentioned configurations  self.render()  self.logging("AES algorithm has been imported.\n")    我们来一行一行代码进行分析：  首先项目定义了一个类UserDefinedWidget，同时传递了一个参数CryptographyWidget。  class UserDefinedWidget(CryptographyWidget)  观察前面导入的包可以知道，CryptographyWidget就是主窗口类，因为每个ui文件都使用到了这个类，所以这个类是我们重点要实现的内容。  同时最终我们要实现的是开一个主窗口，通过点击菜单进行切换，所以主窗口也是必不可少的。  然后是类的初始化方法：  def \_\_init\_\_(self, parent):  CryptographyWidget.\_\_init\_\_(self, parent)  在子类的\_\_init\_\_方法中，使用super()函数调用了CryptographyWidget类的\_\_init\_\_方法。确保在子类的\_\_init\_\_方法中的任何额外代码之前，先执行父类的初始化代码。同时给父窗口传递了一个参数parent，由于没有父窗口的代码，所以我不太理解这个参数的作用。因此最后的实现中将这个参数删除。  然后是对类的属性的定义：  self.setWindowTitle("AES") self.tabs\_config = [IntroductionTab(  link="file:///" + Path.MENU\_DIRECTORY + "/CryptographicAlgorithm/BlockCipherAlgorithm/AES/html/index.html"),  IntermediateValueTab(), SmartCardTab()] # set smart card widget configurations self.smart\_card\_config = SmartCard()  setWindowTitle()方法：设置窗口标题。  传递一个字符串，将窗口标题设置成这个字符串。  效果如下：    tabs\_config属性，观察属性的值可以发现这是一个link链接，将子窗口与html页面联系起来。因为本项目最终的实现不需要web页面，所以考虑将这个属性删除，同时这个属性用到了CryptographyModule包的IntroductionTab  ，IntermediateValueTab，SmartCardTab类，因此这几个类可以不用实现。  smart\_card\_config属性，同上，因为最终的项目要脱离软件的限制，所以智能卡的使用应该从项目中删除。以及涉及到的算法实现也一样需要删除。  # set groups configurations # set plain text edit component configurations # set button component configurations' # id: the identity of the component # clicked\_function: execute the function after the button clicked self.groups\_config = [  Group(name="Key",  plain\_text\_edits=[Key(enabled=True, id="Key", label="Key (Hex)",  default\_text="2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C")],  buttons=[  Button(id="KeyGenerate", name="KeyGenerate", clicked\_function=self.generate\_key)  ]),  Group(name="Encrypt",  plain\_text\_edits=[PlainTextEdit(id="Plaintext", label="Plaintext (Hex)",  default\_text="32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34"),  PlainTextEdit(id="\_Ciphertext", label="Ciphertext (Hex)",  default\_text="", read\_only=True)],  buttons=[  Button(id="ComputerEncrypt", name="Encrypt (PC)", clicked\_function=self.computer\_encrypt),  Button(id="CardEncrypt", name="Encrypt (Card)", clicked\_function=self.card\_encrypt),  Button(id="CleanEncrypt", name="Clean", clicked\_function=self.encrypt\_clean)  ]),  Group(name="Decrypt",  plain\_text\_edits=[PlainTextEdit(id="Ciphertext", label="Ciphertext (Hex)",  default\_text=""),  PlainTextEdit(id="\_Plaintext", label="Plaintext (Hex)",  default\_text="", read\_only=True)],  buttons=[  Button(id="ComputerDecrypt", name="Decrypt (PC)", clicked\_function=self.computer\_decrypt),  Button(id="CardDecrypt", name="Decrypt (Card)", clicked\_function=self.card\_decrypt),  Button(id="CleanDecrypt", name="Clean", clicked\_function=self.decrypt\_clean)  ]) ]  然后是最重要的部分，groups\_config属性：  这个属性包含了CryptographyModule包的Button, PlainTextEdit, Group, ErrorType，KeyGroup，Key这些类。  同时可以观察到实例化这些类时输入的参数。作为实现的考量。  # render user interface based on above-mentioned configurations self.render() self.logging("AES algorithm has been imported.\n")  self.render() 用于渲染用户界面，负责将界面元素和配置组件进行渲染，以展示给用户。  原有的render()方法应该在父类CryptographyWidget里实现，其他的属性也应该在父类中有定义，这里最好的实现就是只调用这样一个渲染函数将窗口实例化。但是每个子窗口的groups\_config属性都不相同，如果不  KeyGroup：  该类表示一组键值对。它继承自QWidget类。它具有组名称、键编辑（键的输入字段）、组合框和按钮的属性。initUI方法设置布局并向组中添加必要的小部件。  class KeyGroup(QWidget):  def \_\_init\_\_(self, name, key\_edit, combo\_box, buttons):  super().\_\_init\_\_()  self.name = name  self.key\_edit = key\_edit  self.combo\_box = combo\_box  self.buttons = buttons  self.initUI()   def initUI(self):  layout = QVBoxLayout()  for key in self.key\_edit:  label = QLabel(key.label)  edit = QLineEdit(key.text)  if not key.enabled:  edit.setDisabled(True)  layout.addWidget(label)  layout.addWidget(edit)  self.setLayout(layout)  self.setWindowTitle(self.name)  Group：  该类表示一组文本编辑框。它也继承自QWidget类。它具有组名称、普通文本编辑框（纯文本的输入字段）和按钮的属性。initUI方法设置布局并向组中添加必要的小部件。  class Group(QWidget):  def \_\_init\_\_(self, name, plain\_text\_edits, buttons):  super().\_\_init\_\_()  self.name = name  self.plain\_text\_edits = plain\_text\_edits  self.buttons = buttons  self.initUI()   def initUI(self):  layout = QVBoxLayout()  group\_label = QLabel(self.name)  layout.addWidget(group\_label)   for edit in self.plain\_text\_edits:  label = QLabel(edit.label)  text\_edit = QTextEdit(edit.text)  if edit.read\_only:  text\_edit.setReadOnly(True)  layout.addWidget(label)  layout.addWidget(text\_edit)   for button in self.buttons:  button\_widget = QPushButton(button.name)  button\_widget.clicked.connect(button.clicked\_function)  layout.addWidget(button\_widget)   self.setLayout(layout)  Key：  该类表示一个键值对。它具有键ID、标签、默认文本、启用状态和只读状态的属性。  class Key:  def \_\_init\_\_(self, id, label, default\_text, enabled=True, read\_only=False):  self.enabled = enabled  self.id = id  self.label = label  self.text = default\_text  self.read\_only = read\_only  PlainTextEdit：  该类表示一个纯文本输入框。它具有输入ID、标签、默认文本和只读状态的属性。它还具有获取和设置输入框文本的方法。  class PlainTextEdit:  def \_\_init\_\_(self, id, label, default\_text, read\_only=False):  self.id = id  self.label = label  self.text = default\_text  self.read\_only = read\_only    def get\_text(self):  return self.text   def set\_text(self, text):  if not self.read\_only:  self.text = text  Button：  该类表示一个按钮。它具有按钮ID、名称和在按钮被点击时执行的函数的属性。  class Button:  def \_\_init\_\_(self, id, name, clicked\_function):  self.id = id  self.name = name  self.clicked\_function = clicked\_function  ComboBox：  该类表示一个组合框，即下拉列表。它具有启用状态、ID、标签、项目（下拉列表中的选项）和当组合框的选择发生变化时执行的函数的属性。它还具有获取和设置组合框的ID、标签和项目的方法。  class ComboBox:  def \_\_init\_\_(self, enabled, id, label, items, changed\_function=None):  self.enabled = enabled  self.id = id  self.label = label  self.items = items  self.changed\_function = changed\_function    def get\_id(self):  return self.id   def set\_id(self, id):  self.id = id   def get\_label(self):  return self.label   def set\_label(self, label):  self.label = label   def get\_items(self):  return self.items   def set\_items(self, items):  self.items = items  Error：  该类表示一个错误。它具有一个错误值的属性。  ErrorType：  这是一个类似枚举的类，它将不同的错误类型定义为Error类的实例。它为SmartCardConnectError、NotMeetRequirementError、CharacterError和LengthError提供预定义的错误值。  class Error:  def \_\_init\_\_(self, value):  self.value = value class ErrorType:  SmartCardConnectError = Error("SmartCardConnectError")  NotMeetRequirementError = Error("NotMeetRequirementError")  CharacterError = Error("CharacterError")  LengthError = Error("LengthError")  Logging：  该类提供日志记录功能。它具有日志小部件（LoggingWidget的实例）的属性。它具有记录消息和记录错误的方法，这些方法将消息附加到日志小部件。  LoggingWidget：  该类表示用于显示日志消息的小部件。它继承自QWidget类。它具有一个QTextEdit小部件用于显示日志消息。append\_log\_message方法将日志消息附加到小部件，append\_error\_message方法将带有特定格式的错误消息附加到小部件。  class Logging:  def \_\_init\_\_(self, log\_widget):  self.log\_widget = log\_widget   def log(self, message):  self.log\_widget.append\_log\_message(message)   def log\_error(self, error):  error\_message = str(error)  self.log\_widget.append\_error\_message(error\_message)  class LoggingWidget(QWidget):  def \_\_init\_\_(self):  super().\_\_init\_\_()   self.initUI()   def initUI(self):  self.setWindowTitle("Logging Widget")  self.setGeometry(300, 300, 500, 400)   self.log\_text\_edit = QTextEdit(self)  self.log\_text\_edit.setReadOnly(True) # 设置为只读模式  self.log\_text\_edit.setPlaceholderText("Log messages will be displayed here.")   layout = QVBoxLayout(self)  layout.addWidget(self.log\_text\_edit)   def append\_log\_message(self, message):  self.log\_text\_edit.append(message)  def append\_error\_message(self, error\_message):  error\_prefix = "<span style='color: red;'>Error: </span>"  formatted\_message = f"{error\_prefix}{error\_message}"  self.log\_text\_edit.append(formatted\_message)  TextEdit：  该类是QTextEdit的子类，并提供了额外的方法来获取和设置文本编辑框的文本。  class TextEdit(QTextEdit):  def \_\_init\_\_(self, parent=None):  super().\_\_init\_\_(parent)   def get\_text(self):  text = self.toPlainText()  return text   def set\_text(self,str\_data):  return self.setText(str\_data)  主窗口：  class CryptographyWidget(QMainWindow):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.logging\_widget = LoggingWidget()  self.logging = Logging(self.logging\_widget)  self.initUI()  self.current\_subwidget = None   def initUI(self):  # 延迟导入  import ClassicCrypto  import BlockCipher  import PublicKeyCryptography  import StreamCipher  import Hash  # import CryptographicProtocol  import MathematicalBasis   self.setWindowTitle("Menu Bar")  self.setGeometry(300, 300, 500, 400)   menubar = self.menuBar()   classic\_cipher\_menu = menubar.addMenu("Classic Cipher")   hill\_cipher\_action = QAction("Hill Cipher", self)  hill\_cipher\_action.triggered.connect(lambda: self.handleCipherAction(ClassicCrypto.HillWidget)) # 修改为通用的处理方法  classic\_cipher\_menu.addAction(hill\_cipher\_action)   caesar\_cipher\_action = QAction("Caesar Cipher", self)  caesar\_cipher\_action.triggered.connect(lambda: self.handleCipherAction(ClassicCrypto.CaesarWidget)) # 修改为通用的处理方法  classic\_cipher\_menu.addAction(caesar\_cipher\_action)  # 因为菜单栏比较多，这里就省略后面的了  #绘制主窗口  self.central\_widget = QWidget(self)  self.setCentralWidget(self.central\_widget)  self.layout = QVBoxLayout(self.central\_widget)  self.layout.setContentsMargins(0, 0, 0, 0)  self.layout.setSpacing(0)  self.show()  方法：  def logging\_error(self, error):  self.logging.log\_error(error)  def pop\_message\_box(self, message):  QMessageBox.critical(self, "Error", message)  def handleCipherAction(self, widget\_class):  widget = widget\_class() # 创建子窗口实例  self.setCentralWidget(widget) # 设置子窗口为中央部件  def closeEvent(self, event):  if hasattr(self, 'current\_widget') and isinstance(self.current\_widget, QWidget):  self.current\_widget.close()  event.accept()  通过一系列密码算法的独立编码接口封装实现（可借助于开源代码），熟悉并掌握常规的分组对称密码算法、序列对称密码算法、HASH函数、MAC函数、数字签名算法的接口编制方法，并能够利用可用编程接口针对文件加解密实现一个实用工具设计，要求体现：  （1）各种密码技术在方案设计中得到正确、合理、合适的集成应用；  （2）方案设计体现安全性与方便性的结合与平衡。  （3）可以针对任意类型、任意大小的文件操作。 | | | |
| 结论分析与体会：  *实验实践设计的先进性与突出特色*  *实验实践中遇到的问题和解决的方法*  *实验结果达到设计目标的程度*  *还可以进行哪些改进*  *实验得到哪些收获和启发*  *……* | | | |

注：斜体灰色文本仅用于提示，填写内容时请删除。

提交时间与方式：按照实验实践任课老师的具体通知。