# 椭圆曲线加密体制的原理与实现分析

作者名

**摘要** 本文介绍了椭圆曲线密码体制以及在该体制的基础上运行的Elgamal加密算法。着重从数学原理、关键算法、具体实现、性能分析和安全性分析的角度对它们进行讨论与分析。

**关键词** 椭圆曲线密码体制 Elgamal加密算法 有限域

# 1.引言（可以介绍该算法的历史和研究现状）

随着计算机网络和通讯技术的迅猛发展和广泛应用，通过网络传输的数据量越来越大，社会的信息化程度越来越高。网络和通讯技术的发展给人们的生活带来了许多好处和巨大的遍历，但是网络安全问题也在日益变得突出。没有网络安全就没有国家安全，可见网络安全不仅对个人有着重要的影响，而且对国家安全社会稳定、经济有秩序的发展都起到了至关重要的作用。

用密码技术保障信息安全是密码最基本最原始的功能，也是信息安全的基础。随着计算机技术的不断发展，密码学也在不断发展以满足人们对安全可靠性的需求。在众多密码技术中，根据研究表明，椭圆曲线密码160bits长的密钥所具有的安全性与RSA中1024bits长的密钥所具有的安全性相当，因此椭圆曲线以其密钥长度短，单位比特安全强度高，灵活性高等优点，掀起了国内外密码学学者的研究热情。

在该篇论文中，我们会对椭圆曲线的算法原理进行介绍，给读者一个初步的映像，同时我们还会讨论关于椭圆曲线算法中关键算法是如何选择和设计的，在讨论完以上内容后我们会通过代码的方式向读者形象生动的展现出椭圆曲线加密算法的实现过程，最后我们会对椭圆曲线算法的安全性进行讨论，分析椭圆曲线算法目前面临的威胁有哪些以及有何种方式可以解决。

# 2.算法原理

## 2.1椭圆曲线

椭圆曲线并非椭圆,之所以称为椭圆曲线是因为它的曲线方程与计算椭圆周长的方程相似。一般的，椭圆曲线指的是由维尔斯特拉斯(Weierstrass)方程

所确定的曲线,它是由方程的全体解 再加上一个无穷远点构成的集合，其中是满足一些简单条件的实数，和也在实数集上取值。上述曲线方程可以通过坐标变换转化为下述形式：

由它确定的椭圆曲线常记为，简记为。

## 2.2素域上的椭圆曲线

设是大于3的素数，且 ,称, 其中 为上的椭圆曲线。该椭圆曲线只有有限个点，其个数由Hasse定理确定。

其中Hasse定理为是有限域上的椭圆曲线，是上点的个数，则

由此可得到一个同余方程：。其解为一个二元组，，将此二元组描画到椭圆曲线上便为一个点，故称其为一个解点。

为利用解点构成交换群，需引进一个0元素，并定义如下加法运算：

1. **定义单位元**

引进一个无穷远点，简记为，作为0元素。

并定义对于所有点

。

1. **定义逆元素**

设和是解点，若 且 ，则。这说明任何解点的逆就是，注意到。

注：规定无穷远点的逆就是其自己。 。

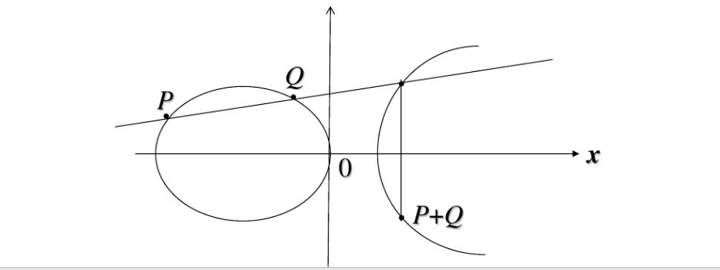
1. **定义加法**

设，且和不互逆，则

。其中

式中： 。

1. **椭圆曲线解点加法运算的几何意义**

设和是椭圆曲线上的两个点，则连接点和 的直线与椭圆曲线的另一个交点关于横轴的对称点即为的点。同时为计算的两倍，在点作一条切线并找到与椭圆曲线的另一个交点，则。

1. **基于的离散对数问题**

（1）设为素数，则模的剩余构成有限域： ， 的非零元素构成乘法循环群，则称为

的生成元。

（2）求的模幂运算为： ,

（3）求对数的运算为 由于上述运算是定义在有限域上的，所以称为离散对数运算。从计算是容易的。可是从计算就困难得多，利用目前最好的算法，对于认真选择的，求解离散对数问题的计算复杂性为。据此对于认真选择的、足够大的，求解离散对数问题是困难的。

设是椭圆曲线上的一个循环子群，是的一个生成元，。已知和，求满足的整数, ，称为椭圆曲线上的离散对数问题。计算的过程称为点乘运算。

1. **椭圆曲线上的Elgamal加密体制**

椭圆曲线上的上的点的阶是指满足的最小正整数，记为，其中是无穷远点。

在使用椭圆曲线密码体制时,首先需要需要选取一条椭圆曲线，并得，将明文消息通过编码嵌入到椭圆曲线上得到点,然后再对点做加密变换，在解密后还得将逆向译码才能获得明文。

（1）密钥生成

在椭圆曲线 上选取一个阶为（为一个大素数）的生成元。随机选取整数，计算。公钥为，私钥为。

（2）加密

为加密为，随机选取一个整数，，计算*，* ，则密文为。

（3）解密

为解密一个密文，计算

攻击者要想从计算出，就必须知道。而要从和中计算出将面临求解椭圆曲线上的离散对数问题。

# 3.关键算法的选择与设计

## 3.1字符到点的映射算法

设椭圆曲线方程为

其中为素数。下面将数字明文植入一个点的的坐标。

算法1 横坐标映射算法

输入：椭圆曲线的参数*，*其中为大于3的素数。数字明文。

输出：映射点的横坐标。

1. 随机取一个大整数，满足。赋初值。
2. ，若，则跳转到步骤1。设，计算。
3. 计算。若，则是模的二次剩余，算法终止，输出；若，则不是模的二次剩余，跳转到步骤2。

根据定理“若是素数，则素数域中，二次剩余元素个数和二次非剩余元素个数相同，各占元素总数的一半”，则对于每一个*，*计算失败的概率为。下面计算给定的点的值。

算法2 纵坐标计算算法（Tonelli-Shanks算法）

输入：椭圆曲线的参数，其中为大于3的素数。椭圆曲线点的横坐标。

输出：椭圆曲线点的纵坐标。

1. 计算。
2. 用2反复除，直至余数为，设此时的商为。设，则有，其中为奇数。若，则，则算法终止，返回。
3. 随机选择一个数，其中是模的非二次剩余。计算
4. 若，算法终止，返回。否则，找出最小的，满足（可以用重复平方进行）。
5. 计算

跳转至步骤4。

下面介绍已知点，计算其对应的明文的算法。

算法3 明文恢复算法

输入：椭圆曲线的参数，其中为大于3的素数。椭圆曲线上一点。

输出：点对应明文。

1. 返回。

## 3.2同时多点乘算法

在椭圆曲线上计算时，我们运用一种用于加速的计算的算法，即同时多点乘算法。设和在二进制下是位数，则两者都可以表示成为一个有个分量的向量。给定窗口宽度，对于，计算值。在步中的每一步，累加器从值

的表接收次倍点和1次点加，而这些值由移过代表矩阵的窗口来确定。

算法4 同时多点乘算法

输入：窗口宽度，，，。

输出：。

1. 对于所有，计算。
2. 记，，其中每个，是长度为的位串，。
3. *。*
4. 对于从到，重复执行

4.1

4.2 。

1. 返回。

## 3.3乘数的平衡长度2表示式求解算法

乘数的平衡长度2表示式求解算法用于分解算法4中的或。以为例，设，把分解为如下形式

其中和的位数近似为的位数的一半。

算法5 乘数的平衡长度2表示式求解算法

输入：整数。

输出：整数，且。

1. 对输入和，应用扩展的Euclidean算法产生一个方程序列，其中，并且余数且严格递减的。令是使的最大下标。
2. 。
3. 若，则；否则，。
4. 计算和。
5. 计算和。
6. 返回。

## 3.4加速点乘算法

加速点乘算法用于计算，其中。

算法6 加速点乘算法

输入：整数*。*其中。*。*

输出：。

1. 记

其中和的位数近似为的位数的一半。这种表示式称为的平衡长度2表示式。其中利用算法5求出。

1. 记

|  |  |  |
| --- | --- | --- |
|  |  |  |

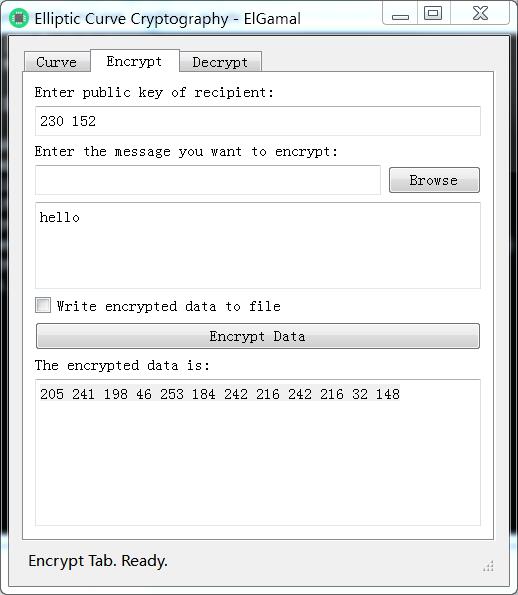
计算，然后利用第1步求出的来赋值式。

因为和的位数近似为的位数的一半，所以有一半的倍点运算被省略。

# 4.算法实现

代码在python2.7，pyqt4环境下编写，输入python gui.py运行程序，当输入错误数据时会报错(一个可行的方案是a=23,b=34,q=277,key=35),当程序返回公钥，即可进行加解密。

范例如下：



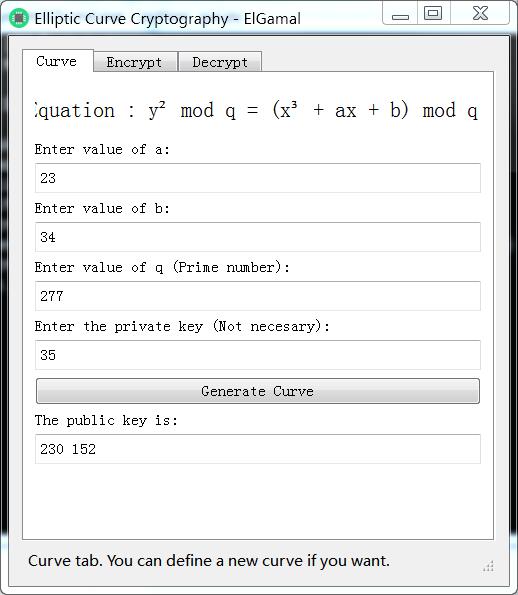


图 1 密钥生成 图 2 加密

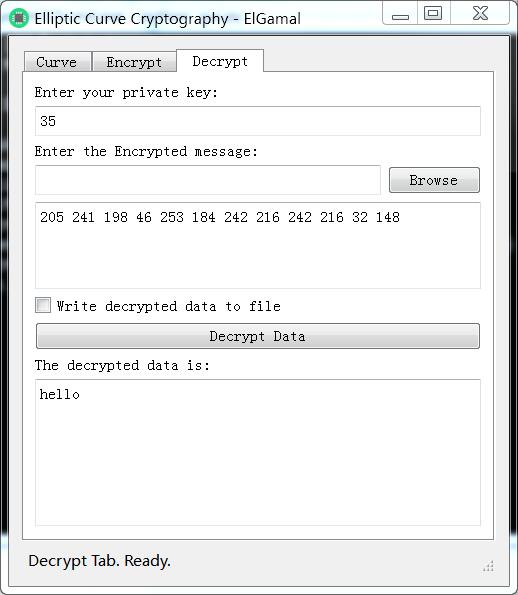


图 3 解密

注：具体代码见附录

# 5.性能分析

## 5.1计算开销小

计算开销是指完成加密和解密过程中所需要的计算量。对DSA 中的运算操作，可以采用预计算的方法来降低计算开销。RSA一般选择 e=65537=(216 +1) 这样e 的二进制表达式中只含两个“1”。 可大大减少计算量。所有应用于离散对数加密系统的预计算技巧同等地应用在基于椭圆曲线的系统中，并且对基于ECC 的密码系统的运算还可以采用快速冗余算法来降低计算开销，快速冗余算法是把连续个“1 ”变成由“0 ”和两个不连续的“1 ”组成，也是减少了“1 ”的个数，从而减少了ECC 算法中点加或数乘的计算开销。各系统的计算开销如表1 所示，表中数据表示完成给定操作所需的时间单元数这些数据没有考虑到各自系统可能采取的一些优化措施，只是提供了大致的比较。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 基于的或(q=160bit) | (n=1024bit，e=216-1) | 离散对数系统（1024bit） |
| 加密 | 120 | 17 | 480 |
| 解密 | 60 | 384 | 240 |
| 签名 | 60 | 384 | 240 |
| 验签 | 120 | 17 | 480 |

表1　各系统的计算开销分析比较表

在现有的计算条件下，对于同等的安全强度，基于ECC 的密码系统在计算能力方面的要求比RSA/DSA 的低很多，所以ECC 密码系统产生密钥的要求比RSA/DSA 密码系统低很多。如ECC 经常被用在计算能力非常有限的智能卡上。另外，由于 ECC 的基域及其元素表示法能被选择，从而域运算(域加、域乘、域求逆) 能被优化。对于基于离散对数和整数因式分解的公钥密码系统就不能做到这一点。

## 5.2计算量小，处理速度快

公钥的生成速度主要是由其中的大数算术运算决定，而大数算术运算的速度跟大数的规模密切相关，在相同计算条件下，ECC 的实现可以选取比RSA 小得多的大数，ECC 的实现速度比RSA 快得多。如表2 所示，在相同安全强度下，ECC 在密钥对的生成、签名以及认证方面，实现速度均比RSA 快得多。如在密钥对生成上，ECC 仅需3.8 ms，而RSA 需要4 708.3 ms，此外，对于ECC 来说，所有应用于离散对数加密系统的计算技巧可以同样应用于基于椭圆曲线的系统中，对于基于ECC 的密码系统的运算还可以采用快速冗余算法来降低计算开销。这使得ECC 在私钥处理速度上快得多。ECC 的计算开销小以及速度快，尤其在存储容量有限且运算能力较低等方面，具有显著优势。实际应用中，在VPN 安全隧道方面，考虑到嵌入式应用在计算机资源和存储资源方面的局限性，根据ECC 加解密速度快、节省带宽、节省存储资源，可选择ECC 来设计和实现身份鉴别；在移动通信以及网络通信方面，需要高效地对数据进行加密，ECC 处理速度快的特点使得移动通信的发展不再受存储容量及低计算能力的限制。此外，ECC 在集成电路卡、数字签名等加密速度要求高的地方能够实现快速、安全的加密和签名。

|  |  |  |
| --- | --- | --- |
| 功能 | 16位 | 1023位 |
| 密钥对生成 | 3.8 | 4708.3 |
| 签名 | 3.0 | 228.4 |
| 认证 | 10.7 | 12.7 |

表2 ECC 密码与RSA 密码软件实现速度的比较

## 5.3存储空间小

ECC 的密钥长度和系统参数与RSA 相比要小得多，由表1 可知，RSA 算法需要512 位，而仅需106 位即可保证其安全性，这就意味着ECC 所需的存储空间小得多。在计算上，所选素数小，计算开销小，这也使得ECC 在存储空间有限制的设备上有更好的应用。随着移动互联网时代的到来，ECC 的这一优势决定了它在移动通信设备、智能卡等存储空间小、运算能力差的设备上的发展地位。智能卡作为电子媒介的主要形式具有体积小、便于携带、安全性高并具一定的数据存储和处理能力、可进行数据加密、解密和数字签名等特点，从而在电子交易中得以广泛的应用。然而，由于其存储容量和计算速度的限制，在对其进行加密时所使用的密钥应尽可能的短，这样才能为智能卡的实用化奠定基础。智能卡的数据传送相对较慢，为提高应用效率，基本数据单元必须小，这样可以减少智能卡与卡的终端之间的数据流量。将ECC 应用于智能卡的优点是生成私钥公钥方便、节省存储空间、节省带宽、提高实用性、节省处理时间，而且不需要增加硬件的处理。ECC 密钥短所带来的优点弥补了智能卡硬件的局限，不仅有效降低了智能卡的生产成本，也提高了实用性。

## 5.4带宽要求低

由于ECC 比其他加密算法密钥短，使得其在传输时带宽要求更低。当对长信息进行加密时，ECC、RSA 密码系统有相同的带宽要求，但应用于短信息时ECC 的带宽要求却低得多，使得ECC 在无线网络中具有广阔的应用前景。目前而言，在无线网络方面，WLAN 的安全问题一直是业界广泛关注的问题，它一直制约着WLAN 的大规模推广及企业级应用。现在使用的SSL 标准安全套接层握手协议带宽开销大而使得网络数据通信效率低，ECC 可减少一定量的带宽开销，使得通信效率得到提高。同时，在Web服务器上的带宽有限使得带宽的费用高昂，而ECC 恰恰解决了Web 服务器的实现遇到的这种瓶颈，节省了计算时间和带宽。在3G 网络方面，针对计算资源和带宽资源有限的弱点，基于ECC 涉及安全的支付流程，可实现端对端的信息安全传输。

# 6.安全性讨论

椭圆曲线加密算法的安全性首先表现在它所确定的椭圆曲线在有限域上的丰富性。由表1可以看出，当素数按位数增大时，椭圆曲线可取得条数大大增加，增加速度远远超过了素数的增加速度。当素数取值为997时，上的椭圆曲线条数为992020条，并且在实际应用中所选取的素数值更大，位数不会低于30位（十进制）。也就是说，在一个确定的有限域上椭圆曲线非常多，完全有条件选取有利于加密计算又不利攻击者反酸的椭圆曲线，对于安全非常有利。

|  |  |  |
| --- | --- | --- |
| 素数位数/位 | 最大素数 | 的条数/条 |
| 1 | 7 | 30 |
| 2 | 97 | 9181 |
| 3 | 997 | 992020 |

表3 参数选取效果比较

椭圆曲线算法的安全性还取决与椭圆曲线离散对数问题（ECDLP）的难解性。设是定义在有限域上的椭圆曲线，点乘，其中、是椭圆曲线上的点，，是椭圆曲线的阶，由给定的和求解，称为椭圆曲线上的离散对数问题。由椭圆曲线的运算规则可知，根据、计算是相当简单的；反之，由、求出在计算上则是非常困难的，这就是椭圆曲线离散对数问题的难解性。而其它常用的公钥密码系统（RSA/DSA）的安全性则是建立在分解两个大素数积的基础之上，解这类问题的通用方法时亚指数时间复杂度的算法。与证书因子分解问题和一般离散对数问题（DLP）不同，目前已知的求解椭圆曲线离散对数问题的最佳算法是Pollard\_rho因子分解算法，它是全指数时间复杂度的算法。因此，一般的用于DLP的有效攻击不能应用到ECDLP上来。从这种意义上讲，椭圆曲线密码系统（ECC）是目前安全性最高的公钥密码系统。

而对于求解ECDLP的最佳算法Pollard\_rho来说它的实际效果也不是那么明显。Pollard\_rho算法的计算需要步，每一步是一个椭圆曲线加运算。假设1MIPS（Million Instructions Per Second每秒百万次指令）机器每秒可以执行次椭圆曲线加运算，则1MIPS机器在一年中能执行的椭圆曲线加运算的次数为：

表2给出了对于有效值使用Pollard\_rho算法计算一个离散对数时计算机的计算能力。一个MIPS年相当于一台计算机固定为1MIPS在一年中的计算能力。

|  |  |  |  |
| --- | --- | --- | --- |
| 域的大小/位 | 的大小/位 |  | MIPS年 |
| 163 | 160 |  |  |
| 191 | 186 |  |  |
| 239 | 234 |  |  |
| 359 | 354 |  |  |
| 431 | 426 |  |  |

表2 参数选取效果比较

例如：10000台1000MIPS的计算机，当时，计算椭圆曲线的离散对数问题需要96000年。据Andrew Odlyzko估计，世界上0.1%的计算机共同工作来破解一密码，在域内计算能力应为MIPS年，在域内计算能力应为MIPS年。在短期安全要求中，至少应该是160位；对于中期的安全要求，至少应是180位。

同时，除了Pollard\_rho算法以外，一些特殊曲线的密码体制也能够较快的实现对ECC系统的有效攻击，如畸形椭圆曲线和超奇异椭圆曲线。过去它们曾经被建议用以构建密码体制，但是目前对这两类特殊曲线的攻击已有了比较有效地方法，如MOV方法和Smart方法。因此，在系统进行椭圆曲线选取时，已禁止使用这几类曲线。

# 7.结论

在该篇论文中，我们主要介绍了椭圆曲线密码体制以及在该体制的基础上运行的Elgamal加密算法。着重从数学原理、关键算法、具体实现、性能分析和安全性分析的角度对它们进行讨论与分析。详细的阐述了椭圆曲线加密算法的各个方面，可以用于读者了解椭圆曲线加密算法使用。通过以上的介绍，我们可以发现，对于相同使用量的参数，椭圆曲线密码体制在每一比特的加密解密过程中都拥有更大的强度，并且所需要的参数规模也较小，因此我们可以预计椭圆曲线密码体制有着无比广阔的应用前景，其应用领域必将不断扩大，成为日后公钥密码发展的方向。当然，如何使得加密算法更加高效应该成为今后研究的重点。目前椭圆曲线加密体制的在软件方面的应用包括SM2、数字签名、指纹识别等，除此之外，椭圆曲线加密算法的硬件实现也有很宽广的发展弓箭，无论是硬件还是软件，寻求快速计算椭圆曲线的新方法都将是椭圆曲线加密算法的一个研究热点。当然，随着量子计算机技术的飞速进步，密码学的发展也在快速进行，如何能够抵挡住量子计算机的运算，目前也是椭圆曲线加密算法的一个研究方向。

# 8.参考文献

[1] 许春香,李发根,聂旭云,禹勇著.现代密码学.电子科技大学出版社. 2008

[2] Darrel Hankerson, Alfred Menezes, Scott Vanstone.椭圆曲线密码学导论.张焕国,等译.电子工业出版社.2005

[3] 侯爱琴，高宝建，辛小龙.信息明文嵌入椭圆曲线的改进算法及实现.计算机应用与软件，2008(7)

# 附录信息

# 分工协作

主要介绍组内分工和个人的完成情况，并对每一个组员进行打分（0~10分）

## gui.py

import os, sys, time

from PyQt4 import QtGui, QtCore

import collections

import encdec

class PopupDialog(QtGui.QDialog):

def \_\_init\_\_(self, parent=None):

QtGui.QWidget.\_\_init\_\_(self)

self.initUI()

def initUI(self):

self.label\_error = QtGui.QLabel()

self.label\_error.setFont(QtGui.QFont('Decorative', 10))

okButton = QtGui.QPushButton("OK")

okButton.clicked.connect(self.close)

center(self)

hori = QtGui.QHBoxLayout()

hori.addStretch(1)

hori.addWidget(self.label\_error)

hori.addStretch(1)

hbox = QtGui.QHBoxLayout()

hbox.addStretch(1)

hbox.addWidget(okButton)

vbox = QtGui.QVBoxLayout()

vbox.addStretch(1)

vbox.addLayout(hori)

vbox.addStretch(1)

vbox.addLayout(hbox)

self.setLayout(vbox)

self.setGeometry(300, 300, 270, 150)

self.setWindowTitle('Error!')

def setval(self, errorval):

self.label\_error.setText(errorval)

class MainWindow(QtGui.QWidget):

def \_\_init\_\_(self):

QtGui.QWidget.\_\_init\_\_(self)

#Window propoerties

self.setGeometry(0,0,500,550)

self.setWindowTitle("Elliptic Curve Cryptography - ElGamal")

self.setWindowIcon(QtGui.QIcon("icon.png"))

self.resize(500,550)

self.setMinimumSize(500,550)

center(self)

#Tabs in the GUI

self.tab\_widget = QtGui.QTabWidget()

#For encryption

tab1 = QtGui.QWidget()

#For decryption

tab2 = QtGui.QWidget()

#For defining curve

tab0 = QtGui.QWidget()

#Layouts for tabs

p0\_vertical = QtGui.QVBoxLayout(tab0)

p1\_vertical = QtGui.QVBoxLayout(tab1)

p2\_vertical = QtGui.QVBoxLayout(tab2)

self.bar = QtGui.QStatusBar(self)

self.bar.showMessage("Ready. Define curve to begin.")

#add tabs, name them

self.tab\_widget.addTab(tab0, "Curve")

self.tab\_widget.addTab(tab1, "Encrypt")

self.tab\_widget.addTab(tab2, "Decrypt")

self.tab\_widget.setTabEnabled(1,False)

self.tab\_widget.setTabEnabled(2,False)

#labels for Tab 1

eqn\_label = QtGui.QLabel("Equation : y\xb2 mod q = (x\xb3 + ax + b) mod q ")

eqn\_label.setMinimumHeight(50)

eqn\_label.setAlignment(QtCore.Qt.AlignCenter)

eqn\_label.setFont(QtGui.QFont('Decorative', 13))

label\_a = QtGui.QLabel("Enter value of a:")

label\_b = QtGui.QLabel("Enter value of b:")

label\_c = QtGui.QLabel("Enter value of q (Prime number):")

label\_priv = QtGui.QLabel("Enter the private key (Not necesary):")

label\_message = QtGui.QLabel("Enter the message you want to encrypt:")

label\_pub = QtGui.QLabel("Enter public key of recipient:")

label\_encrypted = QtGui.QLabel("The encrypted data is:")

label\_decrypted = QtGui.QLabel("The decrypted data is:")

publickey = QtGui.QLabel("The public key is:")

self.write\_encrypted = QtGui.QCheckBox("Write encrypted data to file")

self.write\_decrypted = QtGui.QCheckBox("Write decrypted data to file")

browseButton = QtGui.QPushButton("Browse")

browseButton.clicked.connect(lambda: self.showDialog(1))

self.destTextField = QtGui.QTextEdit()

self.destTextField.setMaximumHeight(label\_a.sizeHint().height()\*2)

self.destTextField.setReadOnly(True)

#Textboxes for Tab 1

self.val\_a = QtGui.QTextEdit()

self.val\_a.setTabChangesFocus(True)

self.val\_b = QtGui.QTextEdit()

self.val\_b.setTabChangesFocus(True)

self.val\_c = QtGui.QTextEdit()

self.val\_c.setTabChangesFocus(True)

self.val\_priv = QtGui.QTextEdit()

self.val\_priv.setTabChangesFocus(True)

self.msg\_val = QtGui.QTextEdit()

self.msg\_val.setTabChangesFocus(True)

self.val\_pub = QtGui.QTextEdit()

self.val\_pub.setTabChangesFocus(True)

self.public = QtGui.QTextEdit()

self.public.setReadOnly(True)

self.encrypted\_string = QtGui.QTextEdit()

self.encrypted\_string.setTabChangesFocus(True)

self.decrypted\_string = QtGui.QTextEdit()

self.decrypted\_string.setTabChangesFocus(True)

#button for curve generation, accompanying Label

button = QtGui.QPushButton("Generate Curve")

button.clicked.connect(self.generateCurve)

button\_enc = QtGui.QPushButton("Encrypt Data")

button\_enc.clicked.connect(self.encryptData)

hori\_box = QtGui.QHBoxLayout()

hori\_box.addWidget(self.destTextField)

hori\_box.addWidget(browseButton)

browseButton2 = QtGui.QPushButton("Browse")

browseButton2.clicked.connect(lambda: self.showDialog(2))

self.destTextField2 = QtGui.QTextEdit()

self.destTextField2.setMaximumHeight(label\_a.sizeHint().height()\*2)

self.destTextField2.setReadOnly(True)

hori\_box2 = QtGui.QHBoxLayout()

hori\_box2.addWidget(self.destTextField2)

hori\_box2.addWidget(browseButton2)

#add elements to Tab 1

p0\_vertical.addWidget(eqn\_label)

p0\_vertical.addWidget(label\_a)

p0\_vertical.addWidget(self.val\_a)

p0\_vertical.addWidget(label\_b)

p0\_vertical.addWidget(self.val\_b)

p0\_vertical.addWidget(label\_c)

p0\_vertical.addWidget(self.val\_c)

p0\_vertical.addWidget(label\_priv)

p0\_vertical.addWidget(self.val\_priv)

p0\_vertical.addWidget(button)

p0\_vertical.addWidget(publickey)

p0\_vertical.addWidget(self.public)

p0\_vertical.addStretch(1)

p1\_vertical.addWidget(label\_pub)

p1\_vertical.addWidget(self.val\_pub)

p1\_vertical.addWidget(label\_message)

p1\_vertical.addLayout(hori\_box)

p1\_vertical.addWidget(self.msg\_val)

p1\_vertical.addWidget(self.write\_encrypted)

p1\_vertical.addWidget(button\_enc)

p1\_vertical.addWidget(label\_encrypted)

p1\_vertical.addWidget(self.encrypted\_string)

#Labels for Tab 2

label\_key = QtGui.QLabel("Enter your private key:")

label\_enc\_message = QtGui.QLabel("Enter the Encrypted message:")

#Buttons for Tab 2

button\_dec = QtGui.QPushButton("Decrypt Data")

button\_dec.clicked.connect(self.decryptData)

#Textboxes for Tab 2

self.priv\_key = QtGui.QTextEdit()

self.priv\_key.setTabChangesFocus(True)

self.encrypted\_data = QtGui.QTextEdit()

self.encrypted\_data.setTabChangesFocus(True)

#Set height for textboxes

self.val\_a.setMaximumHeight(label\_a.sizeHint().height()\*2)

self.val\_b.setMaximumHeight(label\_b.sizeHint().height()\*2)

self.val\_c.setMaximumHeight(label\_c.sizeHint().height()\*2)

self.val\_priv.setMaximumHeight(label\_c.sizeHint().height()\*2)

self.msg\_val.setMaximumHeight(label\_c.sizeHint().height()\*6)

self.public.setMaximumHeight(label\_b.sizeHint().height()\*2)

self.val\_pub.setMaximumHeight(label\_priv.sizeHint().height()\*2)

self.priv\_key.setMaximumHeight(label\_priv.sizeHint().height()\*2)

self.encrypted\_data.setMaximumHeight(label\_priv.sizeHint().height()\*6)

self.encrypted\_string.setMaximumHeight(label\_priv.sizeHint().height()\*20)

self.decrypted\_string.setMaximumHeight(label\_priv.sizeHint().height()\*20)

#Adding widgets to Tab 2

p2\_vertical.addWidget(label\_key)

p2\_vertical.addWidget(self.priv\_key)

p2\_vertical.addWidget(label\_enc\_message)

p2\_vertical.addLayout(hori\_box2)

p2\_vertical.addWidget(self.encrypted\_data)

p2\_vertical.addWidget(self.write\_decrypted)

p2\_vertical.addWidget(button\_dec)

p2\_vertical.addWidget(label\_decrypted)

p2\_vertical.addWidget(self.decrypted\_string)

self.tab\_widget.currentChanged.connect(self.currentTabChanged)

vbox = QtGui.QVBoxLayout()

vbox.addWidget(self.tab\_widget)

vbox.addWidget(self.bar)

self.setLayout(vbox)

def currentTabChanged(self):

if self.tab\_widget.currentIndex() == 1:

self.bar.showMessage("Encrypt Tab. Ready.")

if self.key is not None:

self.val\_pub.setText(self.key)

else:

self.val\_pub.setText("")

self.val\_pub.clearFocus()

elif self.tab\_widget.currentIndex() == 2:

self.bar.showMessage("Decrypt Tab. Ready.")

if self.priv != -1:

self.priv\_key.setText(str(self.priv))

else:

self.priv\_key.setText("")

self.priv\_key.clearFocus()

elif self.tab\_widget.currentIndex() == 0:

self.bar.showMessage("Curve tab. You can define a new curve if you want.")

def showDialog(self, value):

fname = QtGui.QFileDialog.getOpenFileName(self, 'Open file', 'C:')

if value == 1 and fname:

self.destTextField.setPlainText(fname)

self.write\_encrypted.setCheckState(2)

elif value == 2 and fname:

self.destTextField2.setPlainText(fname)

self.write\_decrypted.setCheckState(2)

f = open(fname, 'r')

with f:

data = f.read()

if value == 1:

self.msg\_val.setText(data)

elif value == 2:

self.encrypted\_data.setText(data)

def generateCurve(self):

try:

self.a = int(self.val\_a.toPlainText())

self.b = int(self.val\_b.toPlainText())

self.q = int(self.val\_c.toPlainText())

except:

self.popup("Please input numbers in the necessary fields!")

return

self.priv = int(self.val\_priv.toPlainText() or -1)

self.ed = encdec.Values()

try:

self.key = self.ed.public\_key(self.a, self.b, self.q, self.priv)

except:

self.popup("Not a valid curve! Please define a valid curve.")

if self.key is not None:

self.public.setText(self.key)

else:

self.public.setText("")

self.tab\_widget.setTabEnabled(1,True)

self.tab\_widget.setTabEnabled(2,True)

self.bar.showMessage("Curve defined! Move to Encrypt or Decrypt tabs for more.")

def encryptData(self):

try:

pub\_raw = str(self.val\_pub.toPlainText())

message = self.msg\_val.toPlainText()

except:

self.popup("Something's wrong, I can feel it.")

return

self.bar.showMessage("Encrypting...")

encrypted\_text = self.ed.encryption(pub\_raw, message)

self.encrypted\_string.setText(encrypted\_text)

message\_to\_show = "Encrypted!"

if self.write\_encrypted.isChecked():

path = 'encrypted.txt'

path\_acq = str(self.destTextField.toPlainText())

if path\_acq:

index\_path = path\_acq.rfind(".")

if index\_path != -1:

path = path\_acq[:index\_path] + ".enc." + path\_acq[index\_path+1:]

with open(path, 'w+') as f:

f.write(encrypted\_text)

message\_to\_show = "Encrypted. File saved at " + path

self.bar.showMessage(message\_to\_show)

def decryptData(self):

try:

private\_key = int(self.priv\_key.toPlainText())

cipher\_raw = str(self.encrypted\_data.toPlainText())

except:

self.popup("Error encountered. Please call Vodafone.")

return

self.bar.showMessage("Decrypting...")

finalval = self.ed.decryption(private\_key, cipher\_raw)

self.decrypted\_string.setText(finalval)

message\_to\_show = "Decrypted!"

if self.write\_decrypted.isChecked():

path\_acq = str(self.destTextField2.toPlainText())

path = 'decrypted.txt'

if path\_acq:

index\_path = path\_acq.rfind(".enc")

if index\_path != -1:

path = path\_acq[:index\_path] + ".dec" + path\_acq[index\_path+4:]

else:

index\_path = path\_acq.rfind(".")

if index\_path != -1:

path = path\_acq[:index\_path] + ".dec." + path\_acq[index\_path+1:]

with open(path, 'w+') as f:

f.write(finalval)

message\_to\_show = "Decrypted. File saved at " + path

self.bar.showMessage(message\_to\_show)

def popup(self, errorval):

self.dialog = PopupDialog()

self.dialog.setval(errorval)

self.dialog.show()

#Start window in the center of the screen

def center(widget):

screen = QtGui.QDesktopWidget().screenGeometry()

size = widget.geometry()

widget.move((screen.width()-size.width())/2, (screen.height()-size.height())/2)

app = QtGui.QApplication(sys.argv)

frame = MainWindow()

frame.show()

sys.exit(app.exec\_())

## basicfunc.py

import collections

import os,sys

def inv(n, q):

"""div on PN modulo a/b mod q as a \* inv(b, q) mod q

>>> assert n \* inv(n, q) % q == 1

"""

for i in range(q):

if (n \* i) % q == 1:

return i

pass

assert False, "unreached"

pass

def sqrt(n, q):

"""sqrt on PN modulo: returns two numbers or exception if not exist

>>> assert (sqrt(n, q)[0] \*\* 2) % q == n

>>> assert (sqrt(n, q)[1] \*\* 2) % q == n

"""

assert n < q

for i in range(1, q):

if i \* i % q == n:

return (i, q - i)

pass

return (False, False)

Coord = collections.namedtuple("Coord", ["x", "y"])

## elgamal.py

import basicfunc

class ElGamal(object):

"""ElGamal Encryption

pub key encryption as replacing (mulmod, powmod) to (ec.add, ec.mul)

- ec: elliptic curve

- g: (random) a point on ec

"""

def \_\_init\_\_(self, ec, g):

assert ec.is\_valid(g)

self.ec = ec

self.g = g

self.n = ec.order(g)

pass

def gen(self, priv, g):

"""generate pub key

- priv: priv key as (random) int < ec.q

- returns: pub key as points on ec

"""

return self.ec.mul(g, priv)

def enc(self, plain, pub, g, r):

"""encrypt

- plain: data as a point on ec

- pub: pub key as points on ec

- r: randam int < ec.q

- returns: (cipher1, ciper2) as points on ec

"""

assert self.ec.is\_valid(plain)

assert self.ec.is\_valid(pub)

return (self.ec.mul(g, r), self.ec.add(plain, self.ec.mul(pub, r)))

def dec(self, cipher, priv, ec):

"""decrypt

- chiper: (chiper1, chiper2) as points on ec

- priv: private key as int < ec.q

- returns: plain as a point on ec

"""

c1, c2 = cipher

assert self.ec.is\_valid(c1) and ec.is\_valid(c2)

return self.ec.add(c2, self.ec.neg(self.ec.mul(c1, priv)))

pass

## elliptic.py

import os,sys

import basicfunc

class EC(object):

"""System of Elliptic Curve"""

def \_\_init\_\_(self, a, b, q):

"""elliptic curve as: (y\*\*2 = x\*\*3 + a \* x + b) mod q

- a, b: params of curve formula

- q: prime number

"""

assert 0 < a and a < q and 0 < b and b < q and q > 2

assert (4 \* (a \*\* 3) + 27 \* (b \*\* 2)) % q != 0

self.a = a

self.b = b

self.q = q

# just as unique ZERO value representation for "add": (not on curve)

self.zero = basicfunc.Coord(0, 0)

pass

def is\_valid(self, p):

if p == self.zero: return True

l = (p.y \*\* 2) % self.q

r = ((p.x \*\* 3) + self.a \* p.x + self.b) % self.q

return l == r

def at(self, x):

"""find points on curve at x

- x: int < q

- returns: ((x, y), (x,-y)) or not found exception

>>> a, ma = ec.at(x)

>>> assert a.x == ma.x and a.x == x

>>> assert a.x == ma.x and a.x == x

>>> assert ec.neg(a) == ma

>>> assert ec.is\_valid(a) and ec.is\_valid(ma)

"""

assert x < self.q

ysq = (x \*\* 3 + self.a \* x + self.b) % self.q

y, my = basicfunc.sqrt(ysq, self.q)

if y is not False:

return basicfunc.Coord(x, y), basicfunc.Coord(x, my)

else:

return False, False

def neg(self, p):

"""negate p

>>> assert ec.is\_valid(ec.neg(p))

"""

return basicfunc.Coord(p.x, -p.y % self.q)

def add(self, p1, p2):

"""<add> of elliptic curve: negate of 3rd cross point of (p1,p2) line

>>> d = ec.add(a, b)

>>> assert ec.is\_valid(d)

>>> assert ec.add(d, ec.neg(b)) == a

>>> assert ec.add(a, ec.neg(a)) == ec.zero

>>> assert ec.add(a, b) == ec.add(b, a)

>>> assert ec.add(a, ec.add(b, c)) == ec.add(ec.add(a, b), c)

"""

if p1 == self.zero: return p2

if p2 == self.zero: return p1

if p1.x == p2.x and (p1.y != p2.y or p1.y == 0):

# p1 + -p1 == 0

return self.zero

if p1.x == p2.x:

# p1 + p1: use tangent line of p1 as (p1,p1) line

l = (3 \* p1.x \* p1.x + self.a) \* basicfunc.inv(2 \* p1.y, self.q) % self.q

pass

else:

l = (p2.y - p1.y) \* basicfunc.inv(p2.x - p1.x, self.q) % self.q

pass

x = (l \* l - p1.x - p2.x) % self.q

y = (l \* (p1.x - x) - p1.y) % self.q

return basicfunc.Coord(x, y)

def mul(self, p, n):

"""n times <mul> of elliptic curve

>>> m = ec.mul(p, n)

>>> assert ec.is\_valid(m)

>>> assert ec.mul(p, 0) == ec.zero

"""

r = self.zero

m2 = p

# O(log2(n)) add

while 0 < n:

if n & 1 == 1:

r = self.add(r, m2)

pass

n, m2 = n >> 1, self.add(m2, m2)

pass

# [ref] O(n) add

#for i in range(n):

# r = self.add(r, p)

# pass

return r

def order(self, g):

"""order of point g

>>> o = ec.order(g)

>>> assert ec.is\_valid(a) and ec.mul(a, o) == ec.zero

>>> assert o <= ec.q

"""

assert self.is\_valid(g) and g != self.zero

for i in range(1, self.q + 1):

if self.mul(g, i) == self.zero:

return i

pass

raise Exception("basicfunc.Invalid order")

pass

## encdec.py

import elliptic

import elgamal

import basicfunc

class Values():

def public\_key(self, a, b, q, priv):

self.ec = elliptic.EC(a, b, q)

for i in range(1, q):

self.g, \_ = self.ec.at(i)

if self.g is not False and self.ec.order(self.g) <= self.ec.q and self.ec.order(self.g) > 127:

break

self.eg = elgamal.ElGamal(self.ec, self.g)

self.mapping = [self.ec.mul(self.g, i) for i in range(self.eg.n)]

if priv != -1:

pub = self.eg.gen(priv,self.g)

return\_for\_public = str(pub[0]) + " " + str(pub[1])

return return\_for\_public

else:

return None

def encryption(self, pub\_raw, message):

pub1\_list = pub\_raw.split()

pub1 = int(pub1\_list[0])

pub2 = int(pub1\_list[1])

publ = basicfunc.Coord(pub1,pub2)

mapped = []

for char in message:

mapped.append(self.mapping[ord(str(char))])

cipher = []

for plain in mapped:

cipher.append(self.eg.enc(plain, publ, self.g, 15))

enc\_text = []

for val, single in enumerate(cipher):

if val == 0:

enc\_text.append(str(single[0][0]))

enc\_text.append(str(single[0][1]))

enc\_text.append(str(single[1][0]))

enc\_text.append(str(single[1][1]))

return\_val = " ".join(enc\_text)

return return\_val

def decryption(self, private\_key, cipher\_raw):

cipher\_raw\_list = cipher\_raw.split()

cipher\_super = []

decrypted = []

key = basicfunc.Coord(int(cipher\_raw\_list[0]), int(cipher\_raw\_list[1]))

for i,k in zip(cipher\_raw\_list[2::2], cipher\_raw\_list[3::2]):

cipher\_super.append(basicfunc.Coord(int(i), int(k)))

for cipher in cipher\_super:

decrypted.append(self.eg.dec((key,cipher), private\_key, self.ec))

final\_dec = []

for dec in decrypted:

final\_dec.append(unichr(self.mapping.index(dec)))

return "".join(final\_dec)