南京邮电大学

信息安全综合实验（双语）
实验报告

（ 2021 / 2022 学年第二学期）

题目：基于椭圆曲线签名方案的实现

专　　业	信息安全
组长 学号姓名	B19031614 任远哲
组员 学号姓名	B19022101 王润青
	B15070512 王剑樵
指 导 教 师	李琦
指 导 单 位	计算机学院、软件学院、网络空间安全学院
日　　期	2022.06.05

成员分工	
组长（任远哲）	程序的设计建模（代码），撰写报告相应部分
组员（王润青）	程序的需求建模（基于场景，基于行为），撰写报告相应部分
组员（王剑樵）	软件测试（单元测试，集成测试，确认测试），撰写报告相应部分

教师评价

课程目标	评价准则（每项 10 分）	团队成员		
课程目标 1：通过本次综合实验，培养学生综合应用密码学、计算机技术等领域专业知识的技能，培养学生解决信息安全领域复杂工程问题的实践创新能力。（30 分）	1、能够掌握密码学、计算机等领域的相关基础知识，并能够针对求解的工程问题，进行合理的分析与设计。			
	2、从软件的分析、设计到编制调试，结合计算机网络理论知识、编程语言以及程序设计的方法，能够解决一个和信息安全相关的问题。			
	3、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便。			
课程目标 2：提高学生文献查阅与资料收集能力，发现问题、研究问题、分析问题与解决问题的能力。（20 分）	4、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能。			
	5、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现。			

课程目标 3：培养密码学相关的工程基础实验验证与实现能力，能够根据要求设计实验方案并开展实验，能够对实验数据进行解释与对比分析，给出实验结论。（30 分）	6、能够对实际工程问题进行形式化描述，给出信息安全算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述。			
	7、能够结合计算机软硬件资源，合理选用信息安全算法、数据结构、数据存储方式等技术手段，理解相关算法，对求解的工程问题进行有效建模和求解。			
	8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试。			
课程目标 4：分组完成一次实验设计与开发的全过程，组内成员通过讨论和交流解决实验中的难题，能在实验报告中准确阐述实验内容，能够在答辩时清晰陈述观点和回答问题。（20 分）	9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容的理解，并能够明确团队交流和协作的重要性，以及相关的项目管理知识具有较好的理解和掌握。			
	10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范。			
评价总分				
能力达成评价				
备注： 能力达成评价：优秀、良好、中等、及格、不及格				
指导教师：_____　　_____年___月___日				

基于椭圆曲线签名方案的实现

一、课题内容和要求

椭圆曲线密码体制在执行效率、存储需求等方面要优于 RSA 算法，本课题致力于基于椭圆曲线的签名方案ECDSA的实现。要求密钥长度至少160比特，相关Hash函数选用SHA-1。通过此次实验我们更好的理解椭圆曲线密码体制的实现原理和应用，并提升了编程能力。

题目的具体要求如下：

1.查阅相关资料，深入理解椭圆曲线密码体制的实现思想和原理；

2.选择一种编程语言和开发工具（我们选择了 Python 语言以及 Pycharm 开发环境），编写程序实现椭圆曲线的签名方案，生成并显示公私钥对，利用私钥对指定文本生成签名，利用公钥对签名进行验证，以判定签名的合法性；

3.程序具有图形化用户界面，输出美观；

4.可根据自己能力，在完成以上基本要求后，对程序功能进行适当扩充；

5.撰写报告，对所采用的算法、程序结构和主要函数过程以及关键变量进行详细的说明；对程序的调试过程所遇到的问题进行回顾和分析，对测试和运行结果进行分析；总结软件设计和实习的经验和体会，进一步改进的设想；

6.提供关键程序的清单、源程序及可执行文件和相关的软件说明。

二、需求分析和总体设计

2.1 本课题的主要功能包括：

（1）用户指定一个素数域上的椭圆曲线（即选定 A,B,P），本签名系统可以计算此椭圆曲线上的所有点，选择一个点 G 作为基点并计算它的阶 N；

（2）本签名系统默认使用比特币系统的椭圆曲线签名超参数，但同样支持用户切换成（1）中生成的超参数(A,B,P,G,N)；

（3）用户指定一个私钥，本签名系统可以计算出对应公钥；

（4）用户输入一段待签名的文本和私钥，本签名系统可以根据（2）中的超参数，计算出签名；

（5）用户输入一段待验证的文本和其签名以及对应公钥，本签名系统可以验证此签名的有效性；

绘制用例图，如图 1 所示：

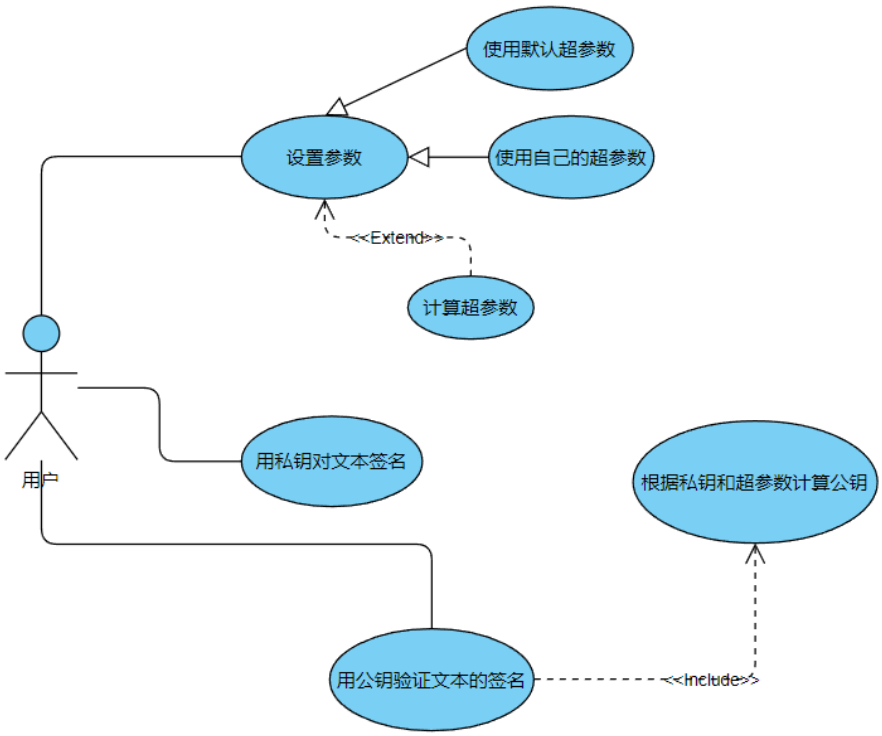

图 1 用例图

2.2 本课题的数据表单设计

本 ECDSA 椭圆曲线签名系统使用的一些参数（设计成全局变量）：

(1)P：一个素数

(2)A，B：素数域上的整数，通过等式 y^2=x^3+A*x+B(mod P)来定义椭圆曲线，如果 A 和 B 取的值不同，那么对应的曲线形状也会不一样

(3)G：椭圆曲线上选择的一个基点（要足够大）

(4)n：点 G 的阶，即 n 是满足 nG=0 的最小正整数。

2.3 本课题的体系结构设计

因为功能并不复杂，所以本系统采用“调用和返回体系结构”风格中的“主程序/子程序体系结构”。 这种的程序结构将功能分解成一个控制层次，其中主程序调用一组程序构件，这些程序构件又去调用别的程序构件。编程是基于面向过程的思想，签名算法模块自己编写，GUI 界面则调用了 Python 中的 Tkinter 库。

2.4 本课题中的主要交互行为建模，绘制序列图，如图 2 所示：

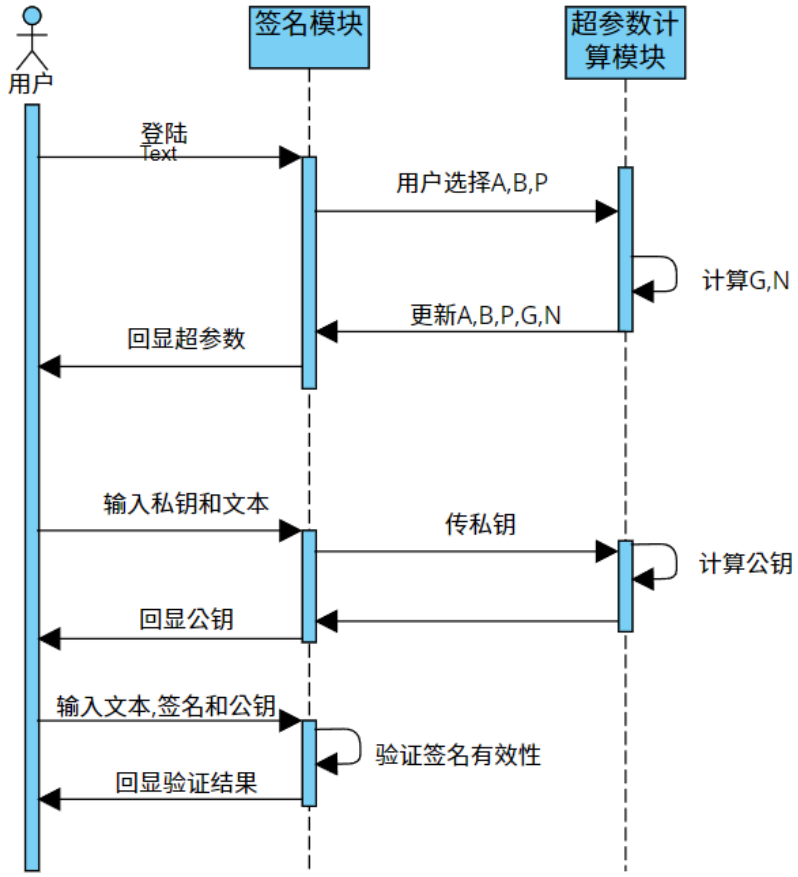

图 2 序列图

2.5 本课题的图形化界面设计

本签名系统总共设计了 5 个子页面。页面 1 为“Generate Keys”，即允许用户输入私钥，然后计算对应公钥（默认使用比特币系统的椭圆曲线参数）；页面 2 为“Sign Text”，即允许用户输入文本和私钥，然后计算签名；页面 3 为“Verify Signature”，即允许用户输入待验证文本，对应签名以及签名者的公钥，然后验证签名的真伪性；页面 4 为“Generate Parameter”，即允许用户自己选择一条椭圆曲线（A,B,P），系统可以自动选择一点 G 并计算它的阶 n；页面 5 为“Settings”，即允许用户重新设置自己的椭圆曲线的参数。

因为篇幅限制，下面仅展示页面 1 的图形化界面，如图 3 所示：

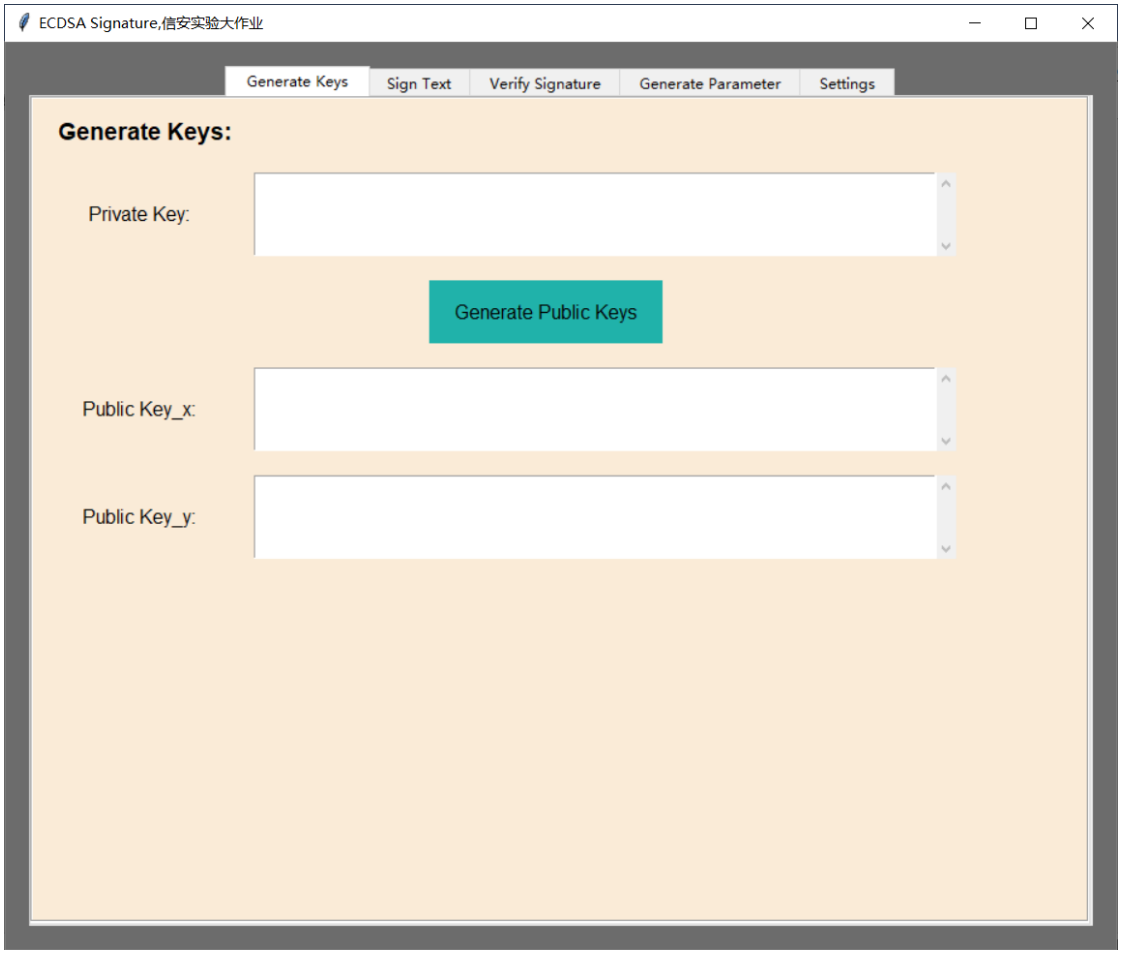

图 3“Generate Keys”界面图

三、椭圆曲线签名算法 ECDSA 的详细设计

椭圆曲线数字签名算法（ECDSA）是使用椭圆曲线密码（ECC）对数字签名算法（DSA）的模拟。ECDSA 于 1999 年成为 ANSI 标准，并于 2000 年成为 IEEE 和 NIST 标准。它在 1998 年既已为 ISO 所接受，并且包含它的其他一些标准亦在 ISO 的考虑之中。与普通的离散对数问题（discrete logarithm problem DLP）和大数分解问题（integer factorization problem IFP）不同，椭圆曲线离散对数问题（elliptic curve discrete logarithm problem ECDLP）没有亚指数时间的解决方法。因此椭圆曲线密码的单位比特强度要高于其他公钥体制。

3.1 椭圆曲线的定义与性质

椭圆曲线其实是一个数学方程，比如下面的方程式:

$$y^2 = (x^3 + a \times x + b)$$

且满足 4a^3 + 27b^2 ≠ 0，这个限定条件是为了保证曲线不包含奇点。

椭圆曲线的图像一般长这样，如图 4 所示：

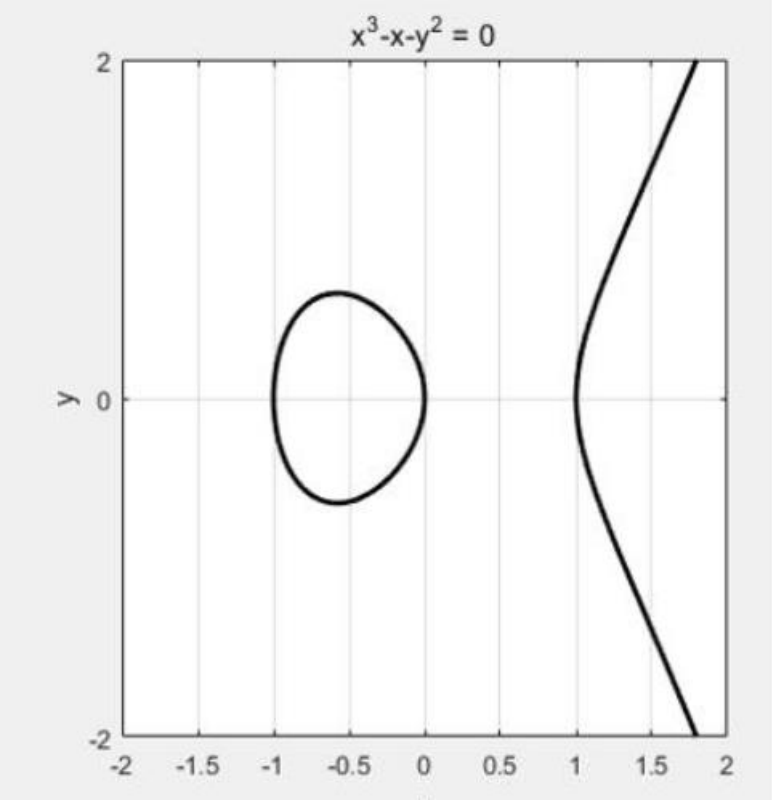

图 4 某椭圆曲线图像

椭圆曲线的图像中，点与点之间好像没有什么联系。能不能建立一个类似于在实数轴上加法的运算法则呢？这就要定义椭圆曲线的加法群，这里需要用到近世代数中阿贝尔群。

在数学中，群是一种代数结构，由一个集合以及一个二元运算所组成。已知集合和运算(G,*)如果是群则必须满足如下要求：

封闭性：∀a,b∈G，a*b ∈ G

结合性：∀a,b,c∈G ，有 (ab)c = a* (b*c)

单位元：∃ e∈G， ∀a ∈G，有 ea = ae = a

逆元： ∀a ∈G ，∃ b∈G 使得 ab = ba = e

阿贝尔群除了上面的性质还满足交换律公理 a * b = b * a

类比一下在椭圆曲线也可以定义阿贝尔群。任意取椭圆曲线上两点 P、Q（若 P、Q 两点重合，则作 P 点的切线），作直线交于椭圆曲线的另一点 R’，过 R’做 y 轴的平行线交于 R，定义 P+Q=R。这样，加法的和也在椭圆曲线上，并同样具备加法的交换律、结合律。如图 5 所示：

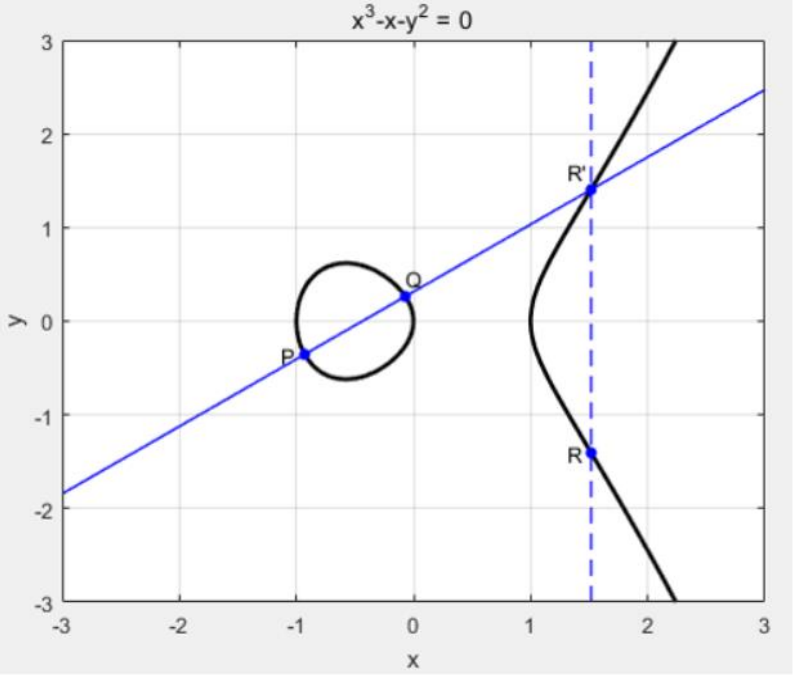

图 5 椭圆曲线上的加法规则

3.2 素数域上的椭圆曲线

椭圆曲线是连续的，并不适合用于加密；所以必须把椭圆曲线变成离散的点，要把椭圆曲线定义在有限域上，有两类：GF(p)上的素数域椭圆曲线和 GF(2^m)上的二次域椭圆曲线。对于 ECDSA 使用素数域 GF(p)上的椭圆曲线。

$$y^2 = x^3 + ax + b \,(\bmod p)$$

此时，已经不是一条完整的椭圆曲线，而是上面全部的整数散点组成的集合，即满足上面方程的所有点(x ,y)再加上一个无穷远点 O 构成的集合,该集合是“有限加法循环群”,上面定义两种运算，点加法(不同的点相加)和点乘法（相同的点相加，特殊的加法）。

一个素数域有如下的性质：

1.GF(p)中有 p（p 为质数）个元素 0,1,2,…, p-2,p-1

2.GF(p)的加法是 a+b≡c(mod p)

3.GF(p)的乘法是 a×b≡c(mod p)

4.GF(p)的除法是 a÷b≡c(mod p)，即 a×b^(-1)≡c (mod p)，b^(-1)也是一个 0 到 p-1 之间的整数，但满足 b×b^(-1)≡1 (mod p)

5.GF(p)的单位元是 1，零元是 0

6.GF(p)域内运算满足交换律、结合律、分配律

类比一下 GF(p)上的椭圆曲线同样有加法，还可以引申出乘法（多次相加）

1.无穷远点 O 是单位元，有 O+ O= O，O+P=P

2.P(x,y)的负元是 (x,-y mod p)= (x,p-y) ，有 P+(-P)= O

3.设 R=P+Q(其中 R 是 PQ 直线与曲线的交点的关于 x 轴的对称点),P(x1,y1),Q(x2,y2)的和 R(x3,y3) 有如下关系：

x3≡kˆ2-x1-x2(mod p)

y3≡k(x1-x3)-y1(mod p)

若 P=Q 则 k=(3x2+a)/2y1 mod p

若 P≠Q，则 k=(y2-y1)/(x2-x1) mod p

4.多个相同的点相加即为椭圆曲线的乘法

3.3 素数域上的椭圆曲线的基点 G

先介绍群的阶和群内元素的阶。群的阶是群内元素个数；群内的一个元素 a 的阶是指会使得 aˆm=e 的最小正整数 m。

假设 a 为循环群的一个生成元（即 a 可以生成循环群内的全部元素）。若给出一个由生成元 a 产生之子群，则满足以下性质：生成元在循环群上的阶=a 生成子群的阶（即循环群内全部元素的个数）

类比到素数域上的椭圆曲线

定义 1 椭圆曲线的阶:椭圆曲线 E,(a, b)在有限域 GF(p)所有离散点的个数，记为 N，称为椭圆曲线的阶。

定义 2 椭圆曲线上点的阶: 点 P=(x,y)∈ E, (a, b),若存在最小的整数 n，使得 nP=0,则称 n 为椭圆曲线上点 P 的阶。

如何选择基点:为了椭圆曲线算法的安全，一般选择椭圆曲线上面阶比较大的一个点作为基点。最好的情况是选择椭圆曲线上可以生成所有点的点都可称为椭圆曲线 E 的基点。也就是说如果椭圆曲线上一点 P 的阶=椭圆曲线的阶,那么它就是最好的椭圆曲线的基点 G。

3.4 基于素数域椭圆曲线的签名算法的可行性

3.4.1.单向陷门性: 椭圆曲线上的乘法 R=k×P。这里的关键在于即使知道了 P 和 R 点，我们也无法计算得到 k。在椭圆曲线算法中没有减法或者除法这种逆向操作。 这是椭圆曲线算法安全性的基础，这个特性也称之为单向陷门性质。我们可以把这个整数 k 作为算法中的私钥，而 R 对应的就是公钥。

3.4.2.准备工作：在使用签名算法之前，需要确定一条椭圆曲线，根据上面知道，如果参数选的不同，那么椭圆曲线方程就会不一样。其中 G 是一个选定的基点，后面所有的运算都会基于这个点开始。也就是说，我们需要选定参数 a，b，p 和 G。n 是 G 的阶，即满足 nG=0 的最小正整数。然后需要定义一对公私钥，根据上面的定义 P=k×G，其中，k 是是从{1,... ,n -1}随机选取的一个正整数 d，其中 n 是 G 的阶，对应为私钥 k0.根据 k0 可以计算出对应为公钥 P0。根据门限函数的性质，知道 P0 很难反推出 k0。

3.5 对文本进行签名的过程

假设要签名的消息是一个字符串 m：“Hello World!”。签名的第一个步骤是对待签名的消息生成一个消息摘要 H(m)，一般采用 SHA1 哈希算法。摘要生成结束后，应用签名算法对摘要 H(m)进行签名：

1.首先从{1,... ,n -1}随机选取的一个正整数 d，其中 **n 是 G 的阶**，对应为私钥 k0.根据 k0 可以计算出对应为公钥 P0。

2.生成一个随机数 k1，注意这个随机数不是上面生成的私钥.。随机数 k1 的范围是[1,n-1]

3.利用 P1=k1×G 计算点 P1，注意这个不是上面的公钥

4.P1 点的横坐标就是 R

5.计算 S = k1^-1 ×(H(m)＋ k0×R)mod p

通常签名的长度是 40 字节，前面 20 字节是 R，后面 20 字节是 S，R 和 S 拼接在一起就是最后的签名。

3.6 对文本以及签名进行验证的过程

验证签名的过程更简单，只需要通过下面的公式:

$$P = S^{-1} \times H \times G + S^{-1} \times R \times P0$$

P0 是公钥，通过这个公式，很容易就是可以计算得到 P，得到 P 之后，如果 P 的横坐标等于 R，那么就验签成功，在验签的过程中，完全不需要私钥的参与。

在上面的算法中，可以发现，除了签名者手上的私钥 k0 之外，在签名的过程中还会生成一个随机数 k1，这个随机数很关键，如果这个数字不够随机，或者使用固定的数字，那么就额可以通过两次签名的使用的哈希值 H1 和 H2 以及 S1 和 S2 来计算这个 k1。如果 k1 知道了，就可以利用上面生成 S 的公式 $S = k1^{-1} \times (H(m) + k0 \times R) \bmod p$ 来计算出私钥 k0，这样一来，私钥就泄漏了。

3.7 椭圆曲线签名算法中参数的选择

为了安全，本签名系统默认使用比特币系统选用的 secp256k1。Secp256k1 为基于 Fp 有限域上的椭圆曲线，由于其特殊构造的特殊性，其优化后的实现比其他曲线性能上可以特高 30%。它的参数如下：

p = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

a = 0

b = 7

G=(0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)

n = 0xFFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

3.8 椭圆曲线签名算法与其他签名算法（如 RSA）的对比

优点

1. 安全性能更高：160 位 ECC 与 1024 位 RSA、DSA 有相同的安全强度
2. 处理速度更快：在私钥的处理速度上，ECC 远 比 RSA、DSA 快得多
3. 带宽要求更低
4. 存储空间更小：ECDSA 的密钥尺寸和系统参数与 RSA 相比要小得多

缺点

1. 设计困难，实现复杂

四、部分核心代码

4.1 GUI 界面模块的代码

4.1.1 主框架绘制代码，设置标题和其下 4 个子页面，如图 6 所示：

```
################################################################################
root = Tk()
root.title("ECDSA Signature,信安实验大作业")
root.geometry("920x750")
# root.configure(bg='white')
s = ttk.Style(root)
s.configure("TNotebook", tabposition='n',background='#6C6C6C')
s.configure("TFrame",background='#FAEBD7')
notebook = ttk.Notebook(root,padding=20);
notebook.pack(expand=1, fill="both")
frame1 = ttk.Frame(notebook,relief=GROOVE,padding=5)
frame2 = ttk.Frame(notebook,relief=GROOVE,padding=5)
frame3 = ttk.Frame(notebook,relief=GROOVE,padding=5)
frame4 = ttk.Frame(notebook,relief=GROOVE,padding=5)
frame5 = ttk.Frame(notebook,relief=GROOVE,padding=5)
notebook.add(frame1, text='   Generate Keys   ')
notebook.add(frame2, text='   Sign Text   ')
notebook.add(frame3, text='   Verify Signature   ')
notebook.add(frame4, text='   Generate Parameter   ')
notebook.add(frame5, text='   Settings   ')
```

图 6 主框架代码

4.1.2 子页面 1 绘制代码。设计输入框输入私钥，计算公钥并回显的页面，如图 7 所示

```
#####################################   page1输入私钥，计算公钥的页面
Label(frame1,text="  Generate Keys: ",font=('Arial', 14, 'bold'),background='#FAEBD7').grid(padx=5,pady=10,row=0,column=0)
#Label(frame1,text="  椭圆曲线参数: ",font=('Arial', 12),background='#FAEBD7').grid(padx=5,pady=10,row=2, column=0)
Label(frame1,text="Private Key: ",font=('Arial', 12),background='#FAEBD7').grid(padx=5,pady=10,row=3,column=0)

privatekey_input=ScrolledText(frame1,width=80,height=5)#私钥输入框
privatekey_input.grid(padx=5,pady=10,row=3, column=1,sticky="E",columnspan=4)

Generate_Keys_Button=Button(frame1,text =" Generate Public Keys ",font=('Arial', 12),activebackground='Coral',background='
Generate_Keys_Button.grid(ipadx=10,ipady=10,padx=5,pady=10,row=4,column=2)

Label(frame1,text="Public Key_x: ",font=('Arial', 12),background='#FAEBD7').grid(padx=5,pady=10,row=5,column=0)
Label(frame1,text="Public Key_y: ",font=('Arial', 12),background='#FAEBD7').grid(padx=5,pady=10,row=6,column=0)
publickey_x_input=ScrolledText(frame1,width=80,height=5)
publickey_x_input.grid(padx=5,pady=10,row=5, column=1,sticky="E",columnspan=4)
publickey_y_input=ScrolledText(frame1,width=80,height=5)
publickey_y_input.grid(padx=5,pady=10,row=6, column=1,sticky="E",columnspan=4)
```

图 7 子页面 1 代码

由于篇幅限制，子页面 2-5 的代码不展示

4.2 超参数定义模块代码

设置超参数 a，b，p，G(BASE_POINT)，N，如图 8 所示：

```
#########################################################################################################
#设置超参数a, b, p,G(BASE_POINT), N,   设置用户的私钥
A = 0
B = 7
# 基点G
BASE_X = 55066263022277343669578718895168534326250603453777594175500187360389116729240
BASE_Y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
BASE_POINT = (BASE_X, BASE_Y)
# 选择的素数P
P = 115792089237316195423570985008687907853269984665640564039457584007908834671663
# 基点G的阶
N = 115792089237316195423570985008687907852837564279074904382605163141518161494337
#########################################################################################################
```

图 8 超参数定义代码

4.3 求乘法逆元模块

4.3.1 拓展欧几里得算法

输入 j 和 k 两个整数，返回(gcd, x, y)，其中 x 和 y 满足 jx + ky = 1，如图 9 所示：

```
def extended_euclidean_algorithm(j, k):#扩展欧几里地算法
    # 扩展欧几里地算法输入j和k两个整数，返回(gcd, x, y)。x和y满足jx + ky = 1
    if j == k:
        return (j, 1, 0)
    else:
        i = 0
        j_array = [j]
        k_array = [k]
        q_array = []
        r_array = []

        prev_r_is_zero = False

        while not (prev_r_is_zero):
            q_array.append(k_array[i]//j_array[i])
            r_array.append(k_array[i]%j_array[i])
            k_array.append(j_array[i])
            j_array.append(r_array[i])
            i += 1
            if r_array[i-1] == 0:
                prev_r_is_zero = True

        i -= 1
        gcd = j_array[i]

        x_array = [1]
        y_array = [0]
```

图 9 拓展欧几里得算法

4.3.2 利用拓展欧几里得算法求逆元

求解(j×x)mod n=1，转化为利用扩展欧几里得，解 j×x+k×n=1，如图 10 所示：

```python
def mod_inverse(j, n):#求乘法逆元
    #求j mod n 的乘法逆元
    (gcd, x, y) = extended_euclidean_algorithm(j, n)

    if gcd == 1:
        return x%n
    else:
        return -1
```

图 10 求乘法逆元

4.4 椭圆曲线上的计算模块

4.4.1 两个不同的点相加

输入两个不同的点（元组类型），返回它们相加的结果（元组类型），如图 11 所示：

```python
def elliptic_add(p, q):#两个不同的点相加
    #输入两个不同的点（元组类型），返回它们相加的结果（元组类型）
    if p == 0 and q == 0: return 0
    elif p == 0: return q
    elif q == 0: return p
    else:
        # Swap p and q if px > qx.
        if p[0] > q[0]:
            temp = p
            p = q
            q = temp
        r = []

        slope = (q[1] - p[1])*mod_inverse(q[0] - p[0], P) % P

        r.append((slope**2 - p[0] - q[0]) % P)
        r.append((slope*(p[0] - r[0]) - p[1]) % P)

        return (r[0], r[1])
```

图 11 不同的点相加

4.4.2 两个相同的点相加，如图 12 所示：

```python
def elliptic_double(p):#两个相同的点相加
    r = []

    slope = (3*p[0]**2 + A)*mod_inverse(2*p[1], P) % P
    r.append((slope**2 - 2*p[0])%P)
    r.append((slope*(p[0] - r[0]) - p[1])%P)

    return (r[0], r[1])
```

图 12 相同的点相加

4.4.3 倍乘

其中 p 是一个点（元组），s 是倍数，返回元组形式的结果，如图 13 所示：

```
def elliptic_multiply(s, p):#倍乘，其中P是一个点（元组），s是倍数

    n = p
    r = 0

    s_binary = bin(s)[2:]
    s_length = len(s_binary)

    for i in reversed(range(s_length)):
        if s_binary[i] == '1':
            r = elliptic_add(r, n)
        n = elliptic_double(n)

    return r
```

图 13 倍乘

4.5 公钥生成模块

输入一个整数作为私钥，计算其公钥（分为两部分，以元组形式返回），如图 14 所示：

```
def generate_public_key():#private_key是一个数字
    #输入一个整数作为私钥，计算其公钥（分为两部分，以元组形式返回）
    private_key=privatekey_input.get(1.0,END)
    private_key=int(private_key)

    publickey_x_input.delete('1.0', END)
    publickey_y_input.delete('1.0', END)
    publickey_x_input.insert(1.0,elliptic_multiply(private_key, BASE_POINT)[0])
    publickey_y_input.insert(1.0, elliptic_multiply(private_key, BASE_POINT)[1])

    return elliptic_multiply(private_key, BASE_POINT)
```

图 14 计算公钥

4.6 基点 G 的生成模块

4.6.1 计算椭圆曲线在域上的点集，如图 15 所示：

```
# 计算椭圆曲线在Fp域上的点集
def cal_points(p, a, b):
    return [(x, y) for x in range(p) for y in range(p) if (y*y - (x*x*x + a*x + b)) % p == 0]
```

图 15 计算点集

4.6.2 计算椭圆曲线点集中一点 p 的阶，如图 16 所示：

```
def get_order(x0,y0,a,b,p):#计算点P(X0,Y0)在椭圆曲线中的阶需要a,p参数
    # 计算-p,逆元：一个椭圆曲线上的点P的逆元，是相对x坐标的对称点
    x1 = x0
    y1 = (-1 * y0) % p
    temp_x = x0
    temp_y = y0
    n = 1
    while True:
        n += 1
        if temp_x==x0 and temp_y==y0:
            p_value = elliptic_double2((temp_x, temp_y),a,p)
        else:
            p_value = elliptic_add2((temp_x, temp_y), (x0, y0), p)

        if p_value[0] == x1 and p_value[1] == y1:
            print("==========该椭圆曲线的阶为%d=========" % (n + 1))
            return n + 1

        temp_x = p_value[0]
        temp_y = p_value[1]
    return -1 #若找不到，则P是无限阶
```

图 16 计算点的阶

4.6.3 挑选一个点作为基点，如图 17 所示：

```
def generate_parameter():#根据a,b,p随机挑选G并计算G的阶N
    a = int(A_choose.get(1.0, END))
    b = int(B_choose.get(1.0, END))
    p = int(P_choose.get(1.0, END))

    Points=cal_points(p, a, b)
    while 1:
        point=random.choice(Points)
        if get_order(point[0],point[1],a,b,p)!=-1:
            BASE_X_Generate.delete('1.0', END)
            BASE_Y_Generate.delete('1.0', END)
            N_Generate.delete('1.0', END)

            BASE_X_Generate.insert(1.0, point[0])
            BASE_Y_Generate.insert(1.0, point[1])
            N_Generate.insert(1.0, get_order(point[0],point[1],a,b,p))
            break
```

图 17 挑选基点

4.7 签名模块

输入私钥和一段文本，生成签名（分为两部分，以元组形式返回），如图 18 所示：

```
def sign():
    #输入私钥和一段文本，生成签名（分为两部分，以元组形式返回）
    private_key=privatekey_input2.get(1.0,END)
    private_key=int(private_key)
    message=msg_input.get(1.0,END)

    hashed_message = sha1(message.encode('utf-8')).hexdigest()#哈希操作针对的是01串.St
    hashed_message=int(hashed_message, 16)#哈希字符串即01串(当做16进制)转整数

    # A secure random number
    k = secrets.randbelow(P)#取值范围是【1, n-1】

    random_point = elliptic_multiply(k, BASE_POINT)

    rx = random_point[0] % N
    signature_proof = mod_inverse(k, N) * (hashed_message + rx*private_key) % N

    sig_rx_input.delete('1.0', END)
    sig_signature_proof_input.delete('1.0', END)
    sig_rx_input.insert(1.0,rx)
    sig_signature_proof_input.insert(1.0,signature_proof)
    return (rx, signature_proof)
```

图 18 签名模块

4.9 签名验证模块

输入公钥（元组形式），签名（元组形式）和文本，返回布尔值，如图 19 所示：

```
def verify():
    #输入公钥（元组形式），签名（元组形式）和文本，返回布尔值
    rx=sig2_rx_input.get(1.0,END)
    rx=int(rx)
    s=sig2_signature_proof_input.get(1.0,END)
    s=int(s)
    public_key_x=publickey_x_input2.get(1.0,END)
    public_key_x=int(public_key_x)
    public_key_y = publickey_y_input2.get(1.0, END)
    public_key_y = int(public_key_y)
    public_key=(public_key_x,public_key_y)

    message=msg2_input.get(1.0,END)

    hashed_message = sha1(message.encode('utf-8')).hexdigest()  # 哈希操
    hashed_message = int(hashed_message, 16)  # 哈希字符串即01串(当做16进制

    inverse_s = mod_inverse(s, N)

    a = elliptic_multiply(hashed_message * inverse_s % N, BASE_POINT)
    b = elliptic_multiply(rx * inverse_s % N, public_key)
    recovered_random_point = elliptic_add(a, b)

    if_verify.delete('1.0', END)
    if_verify.insert(1.0,recovered_random_point[0] == rx)
    return recovered_random_point[0] == rx
```

图 19 签名验证模块

五、软件测试及其结果分析

5.1 单元测试（以计算计算椭圆曲线在域上的点集为例）

```
def cal_points(p, a, b):
    return [(x, y) for x in range(p) for y in range(p) if (y*y - (x*x*x + a*x + b)) % p == 0]
print(cal_points(17,3,5))
```

结果如下：[(1, 3), (1, 14), (2, 6), (2, 11), (4, 8), (4, 9), (5, 3), (5, 14), (6, 1), (6, 16), (9, 8), (9, 9), (10, 7), (10, 10), (11, 3), (11, 14), (12, 1), (12, 16), (15, 5), (15, 12), (16, 1), (16, 16)]

5.2 集成测试

5.2.1 选择私钥，如图 20 所示：

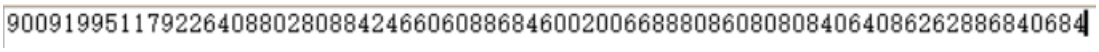

图 20 私钥

5.2.2 计算出的公钥如下，如图 21 所示：

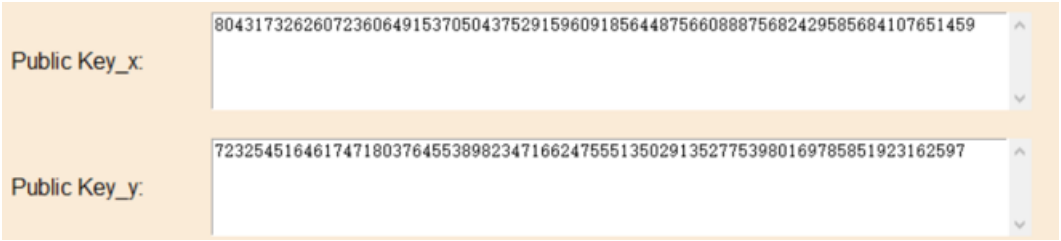

图 21 公钥

5.2.3 对文本“B19031614 任远哲”进行签名，签名结果如图 22 所示：

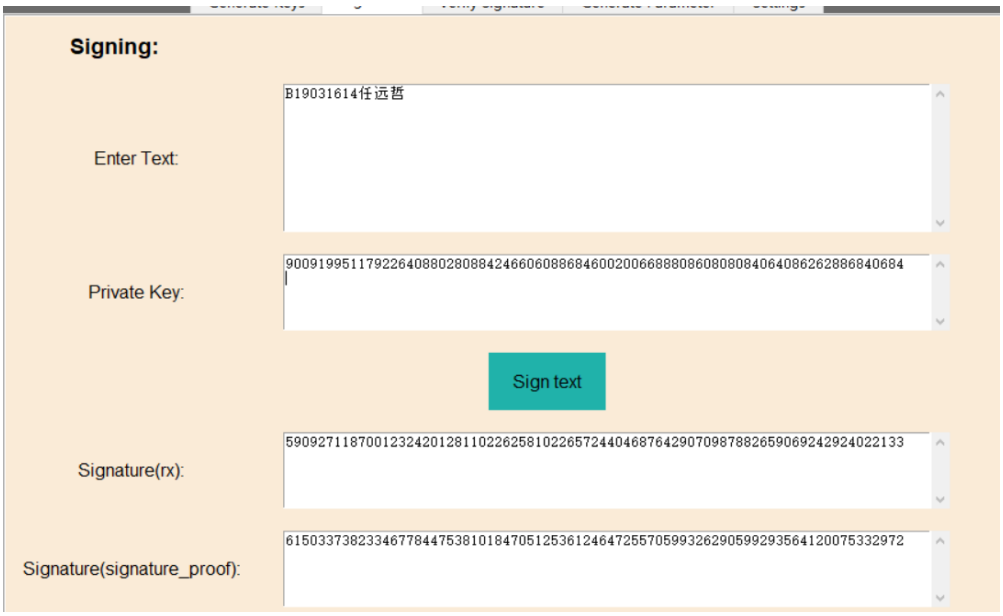

图 22 签名结果

5.2.4 对签名结果进行验证，is_correct 字段为 1 表示通过验证，如图 23 所示：

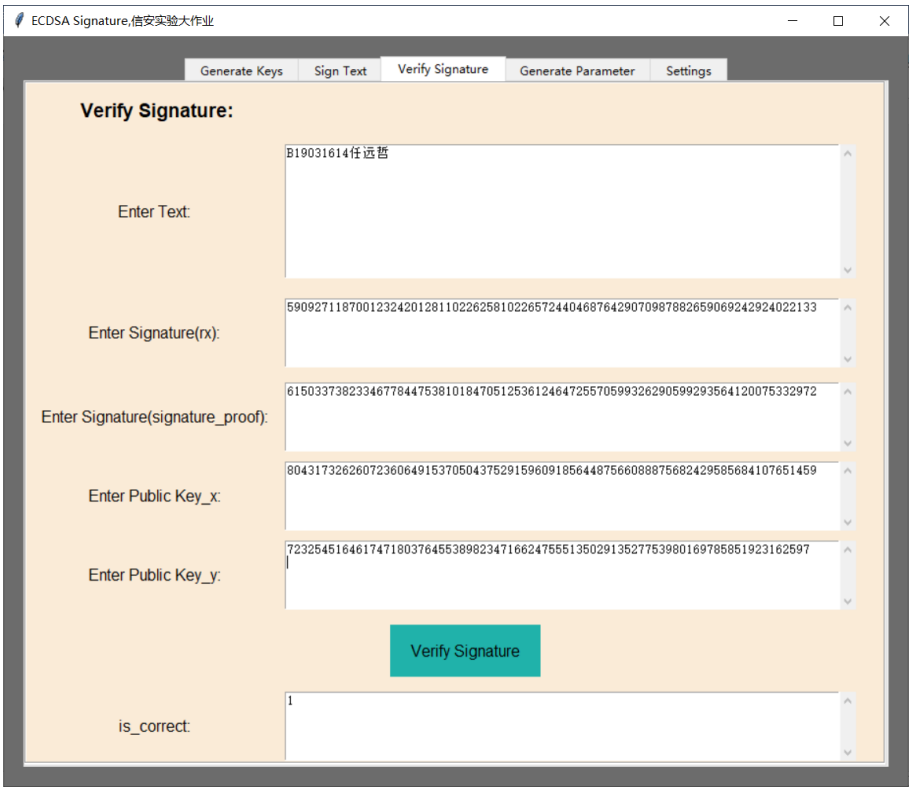

图 23 签名验证结果

5.2.5 略微修改文本内容，再次进行验证。显示没能通过，如图 24 所示。

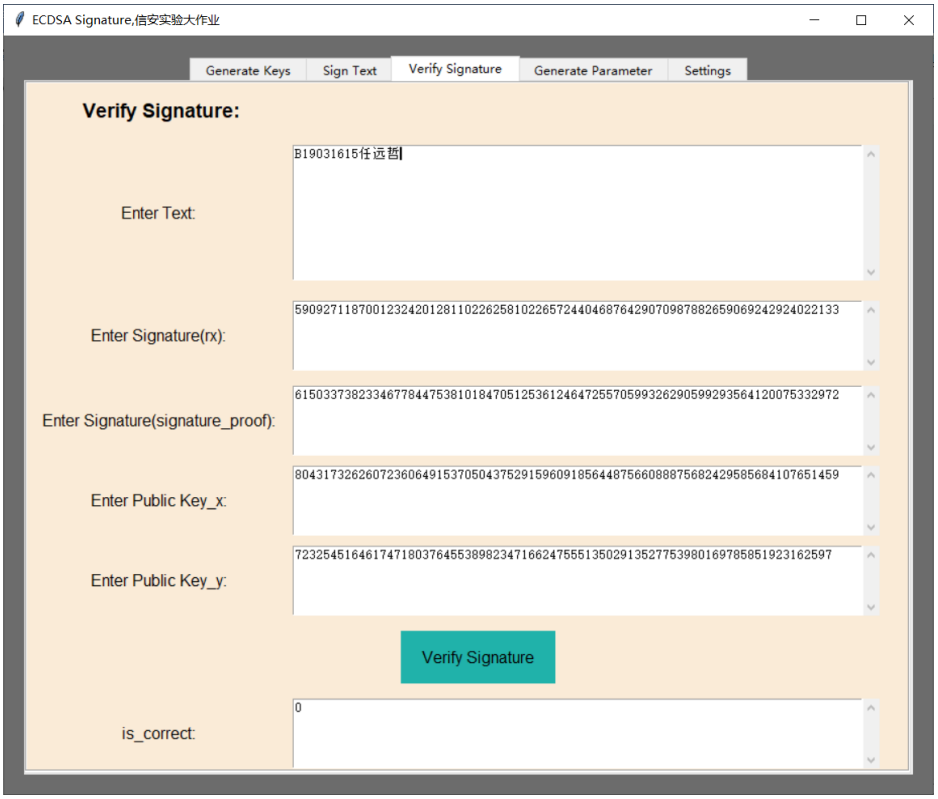

图 24 签名验证结果

六、课题完成过程中遇到的问题及解决方法

问题 1：相比于 RSA 签名，椭圆曲线签名算法更加复杂且难以理解

解决方法：查阅书籍，他人博客，向老师请教

问题 2：Python 语言中函数内部无法修改超参数（全局变量）

解决方法：使用 global 关键字，即可在函数内部访问并修改全局变量，如图 25 所示：

```
def change_hyperparameters(): #修改签名算法中的超参数
    global A
    global B
    global BASE_X
    global BASE_Y
    global BASE_POINT
    global P
    global N

    A = A_change.get(1.0, END)
    A=int(A)
    B = B_change.get(1.0, END)
    B = int(B)
    BASE_X = BASE_X_change.get(1.0, END)
    BASE_X = int(BASE_X)
    BASE_Y = BASE_Y_change.get(1.0, END)
    BASE_Y = int(BASE_Y)
    P = P_change.get(1.0, END)
    P = int(P)
    N = A_change.get(1.0, END)
    N = int(N)
```

图 25 global 关键字

问题 3：软件功能需求随着理解的深入不断增多，开发起来复杂

解决方法：采用增量过程模型。首先实现核心产品功能，也就是签名，验证模块，使用比特币系统默认参数做一个 GUI；后续再扩展功能，比如加上椭圆曲线基点的选择和计算基点的阶的模块。在具体编码时尽量做到高内聚，低耦合的思想。

问题 4：找不到绘制用例图，序列图的免费软件

解决方法：使用在线平台

https://online.visual-paradigm.com/cn/diagrams/features/sequence-diagram-software/

七、总结

通过这次实验，我们学习了基于椭圆曲线签名算法的原理，并动手开发了一个简单的图形化界面程序，这加深了我们对于此算法的理解。数字签名技术有着非常广泛的用途，它可以解决伪造、抵赖和篡改等问题，现在各网站所提供的数字证书，其实质就是 CA 对一个网站的数字签名；数字签名还广泛应用于新兴的区块链技术中，比特币、以太币等数字货币均采用数字签名技术算法保证交易的安全性.简单的说法是：用户利用私钥对交易信息进行签名，并把签名发给矿工，矿工通过验证签名确认交易的有效性。

本课题研究的基于椭圆曲线的数字签名 ECDSA 是所有数字签名中比较完备的一种。基于椭圆曲线的加密算法是在基于离散对数的加密算法基础之上形成的，它的安全性更高，RSA 的破译和求解难度是亚指数级的，而 ECDSA 的破译和求解难度基本上是指数级的。ECDSA 能够凭借 RSA 体制小的参数，提供更高的安全性。图 26 是两者安全级别的对比：

ECDSA 密钥长度/次	RSA 密钥长度/次	攻破时间 MIPS一年/次
160	1024	1012
320	5120	1036
600	21000	1078
1200	120000	10168

图 26 ECDSA 和 RSA 对比

使用较小参数可以提高速度和应用较小的密钥与证书，对于这些优点可以在处理能力、存储空间、带宽和能源消耗表现的很突出，如智能卡和便携式电话机上使用时尤其重要。

但是 ECDSA 也不是万无一失的，比如格攻击算法：2001 年，罗默提出了针对椭圆曲线签名算法的格攻击方法。其中，攻击者借助于其他攻击手段（如利用改变符号的故障而注入攻击）获得临时密钥的部分值（即一部分密钥），再利用求解格上问题获得完整的临时密钥，从而得到真正的私钥。所以，我们在做开发时不能只依赖一种技术，还是要集百家之长。

在实验过程中我们也遇到了很多问题，所幸当下互联网上的资料非常丰富，通过自学我们能解决绝大部分。对于难以解决的问题，我们选择向老师请教，最终完全理解了这个课题。在写开发程序和写实验报告的过程中，我们也复习并实践了软件工程课程中所学知识。比如需求建模，高内聚低耦合的设计思想以及单元测试集成测试等概念。总而言之，通过此课题我们的各方面能力都得到了提升，衷心感谢《信息安全综合实验》这门课程给予我们这样一个实践学习的机会。