**密码学原理**

**实验三：公钥加密**

学号：2022113564 姓名：张哲恺

**实验目的**：本实验旨在让学生掌握运用密码学工具生成RSA密钥，进行非对称加解密，并分析相同因子RSA公钥理解RSA的安全性。

1. **公钥加密**
2. 使用密码学工具实现混合加密过程

要求：生成RSA密钥对（公钥1和私钥1），用公钥1加密对称密钥并采用对称加密方法加密图片，用私钥1解密对称密钥，然后解密图片。（你可以使用实验二中的对称加密方案）

调用rsa.generate\_private（）生成私钥，设定指数为65537，密钥长度为2048，后端选择默认后端。再通过私钥private\_key.public\_key()生成公钥，将私钥和公钥分别保存在rsa\_private\_key.pem和rsa\_public\_key.pem.

Key\_genenrate.py:

1. from cryptography.hazmat.primitives.ciphers.aead import AESGCM

2. from cryptography.hazmat.backends import default\_backend

3. from cryptography.hazmat.primitives import serialization

4. from cryptography.hazmat.primitives.asymmetric import rsa

5. from cryptography.hazmat.primitives import hashes

6. from cryptography.hazmat.primitives.asymmetric import padding

7. import numpy as np

8. from PIL import Image

9. import os

10. import hmac

11.

12. # 生成RSA密钥对

13. private\_key = rsa.generate\_private\_key(

14.     public\_exponent=65537,

15.     key\_size=2048,

16.     backend=default\_backend()

17. )

18.

19. # 获取公钥

20. public\_key = private\_key.public\_key()

21.

22. # 将私钥保存为.pem文件

23. pem\_data\_private = private\_key.private\_bytes(

24.     encoding=serialization.Encoding.PEM,

25.     format=serialization.PrivateFormat.PKCS8,

26.     encryption\_algorithm=serialization.NoEncryption()

27. )

28.

29. with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\rsa\_private\_key.pem", "wb") as pem\_file\_private:

30.     pem\_file\_private.write(pem\_data\_private)

31.

32. print("Private key saved as rsa\_private\_key.pem")

33.

34. # 将公钥保存为.pem文件

35. pem\_data\_public = public\_key.public\_bytes(

36.     encoding=serialization.Encoding.PEM,

37.     format=serialization.PublicFormat.SubjectPublicKeyInfo

38. )

39.

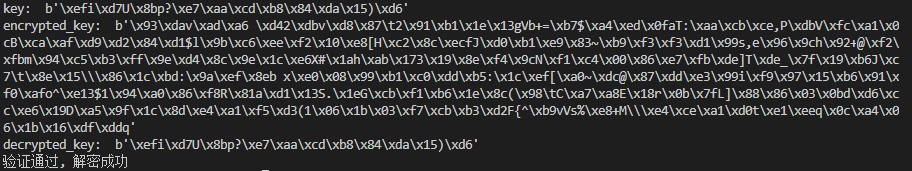
40. with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\rsa\_public\_key.pem", "wb") as pem\_file\_public:

41.     pem\_file\_public.write(pem\_data\_public)

42.

43. print("Public key saved as rsa\_public\_key.pem")

图片加密过程和实验二一样，首先从文件中获取公钥和私钥，加密时使用AES\_GCM加密方法，并且使用公钥加密对称密钥，解密时先用私钥解密对称密钥再解密图像。



RSA.py:

1. from cryptography.hazmat.primitives.ciphers.aead import AESGCM

2. from cryptography.hazmat.backends import default\_backend

3. from cryptography.hazmat.primitives import serialization

4. from cryptography.hazmat.primitives.asymmetric import rsa

5. from cryptography.hazmat.primitives import hashes

6. from cryptography.hazmat.primitives.asymmetric import padding

7. import numpy as np

8. from PIL import Image

9. import os

10. import hmac

11.

12. def encrypt(key) :

13.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\input\_image.bmp" , 'rb') as f:

14.         input\_image = bytearray(f.read())

15.

16.     input\_image[10] = 98 #修改图片数据起始位置

17.

18.     plaintext = np.array(input\_image[55: ]) #记录图片数据

19.     #加密过程

20.     aesgcm = AESGCM(key)

21.     # print("key: " ,  key)

22.     nonce = os.urandom(12)

23.     # print("nonce: " , nonce.hex().upper())

24.     ciphertext = aesgcm.encrypt(nonce , plaintext.tobytes() , None)

25.     hmac\_value = hmac.new(key, ciphertext, digestmod='sha256').digest()

26.     # print("hmac\_value: " , hmac\_value.hex().upper())

27.

28.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\encrypted\_image.bmp" , "wb") as f :

29.         f.write(input\_image[:55]) #文件头

30.         f.write(nonce) #nonce

31.         f.write(hmac\_value) #HMAC消息

32.         f.write(ciphertext) #加密图像数据

33.

34. def decrypt(encrypted\_key) :

35.     key = private\_key.decrypt(

36.         encrypted\_key,

37.         padding=padding.PKCS1v15()

38.     )

39.     print("decrypted\_key: " , key)

40.     aesgcm = AESGCM(key)

41.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\encrypted\_image.bmp" , "rb") as f :

42.         bmphead = bytearray(f.read(55)) #获取文件头

43.         nonce = bytearray(f.read(12)) #nonce

44.         # print(nonce.hex().upper())

45.         hmac\_v = bytearray(f.read(32)) #hmac

46.         # print(hmac\_v.hex().upper())

47.         encrypted\_image = bytearray(f.read())

48.     hmac\_value = hmac.new(key , encrypted\_image , digestmod='sha256').digest()

49.     if (hmac\_v == hmac\_value) :

50.         bmphead[10] = 54 #修改图像数据起始位置

51.         ciphertext = np.array(encrypted\_image)

52.         decrypted\_img = aesgcm.decrypt(nonce , ciphertext.tobytes(), None) #解密

53.         with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\decrypted\_image.bmp" , "wb") as f :

54.             f.write(bmphead)

55.             f.write(decrypted\_img)

56.         print("验证通过, 解密成功")

57.     else :

58.         with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\decrypted\_image.bmp" , "wb") as f :

59.             f.write(bmphead)

60.             f.write(nonce)

61.             f.write(hmac\_v)

62.             f.write(encrypted\_image)

63.         print("验证失败，解密失败")

64.

65. key = AESGCM.generate\_key(bit\_length=128)

66.

67. #加载公钥

68. with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\rsa\_public\_key.pem", "rb") as pem\_file\_public:

69.     public\_key = serialization.load\_pem\_public\_key(

70.         pem\_file\_public.read(),

71.         backend=default\_backend()

72.     )

73.

74. # 加载私钥

75. with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\rsa\_private\_key.pem", "rb") as pem\_file\_private:

76.     private\_key = serialization.load\_pem\_private\_key(

77.         pem\_file\_private.read(),

78.         password=None,  # 如果私钥使用了密码，这里需要提供密码

79.         backend=default\_backend()

80.     )

81.

82. #加密AES密钥

83. encrypted\_key = public\_key.encrypt(

84.     key,

85.     padding=padding.PKCS1v15()

86. )

87. print("key: " , key)

88. print("encrypted\_key: " , encrypted\_key)

89. encrypt(key)

90. decrypt(encrypted\_key)

1. **相同因子公钥分析**
2. 通过分析RSA公钥的因子得到私钥并破解加密信息

要求：分析附件中给出的两个公钥（公钥1、公钥2，均为pem格式）中大整数的公共因子，得到公钥1的私钥，并用私钥解密对称密钥，再解密对称加密的图片。

* 1. 对称加密方法为密钥长度128位的AES-CBC，对明文采用[PKCS #7](https://datatracker.ietf.org/doc/html/rfc2315#section-10.3)填充，128位IV放在密文开头。
  2. 对称加密的明文为RGBA四通道图像中的所有像素，为使密文图片尺寸合法，对密文进行了填充，以四字节（一个像素）为单位，与[PKCS #7](https://datatracker.ietf.org/doc/html/rfc2315#section-10.3)类似，即如果密文图像最后一个像素转换为四字节整数的值为k（大端序），说明密文图像的后k个像素是padding。由此填充方法产生的密文图片比明文图片多一行像素。
  3. 对称加密产生的密文图片为enc1.png
  4. 128位对称密钥先进行Base64编码，再使用公钥1加密，加密方法为RSA-OAEP，密文的Base64编码在下面给出。

|  |
| --- |
| **公钥1：**  -----BEGIN PUBLIC KEY-----  MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAuz20BUTcqVDjzEOKiJF9  66LbQB/59lnXTj/SmiD07mV1XE03BLrWfi7jFh/iq5ZPzVXfbNPjHiojO9WRhWzr  wiQGZNVZ7qFoO/PzXOT8OyHyOMcrb6ogtCyFvDOeximr3M/ICmliU2JxbLSfteZj  AplHJVgs5bJ5LTW7eSy1x2Z5aOsHjesK3rkLi1yB2jM0MeaNIB/Enb82bBMKzAam  vN6tY8bQbEoRbTnlX6PUfkU9w7XsWLMa3QbpIH9mNam1Qz4ynCjWXcDo6KzYotUf  TgGlIIOOJKsAqgOgSHqTz83e8bBizPwJg+CxBzP4Ha8C9phc41i2GiEgDf4J1J0R  0BZDcJEgZIlI+B5tlvJTy/uQyvmEP+hyMD8d83RdzLYy9h8u0MNHjJygY/Kktftp  wPtZPThpMOWWbOMM72a8Y2usz5rKTBAe+bN5QyELCErc/aQB0ABUSsNf4XxaQWbz  gJdb3hEvUkas0PfHui8UB6Yuaa7RmEE6EPIELx2WF2BGw1AG8vg5mi3I+HYxpk9W  mxy2gj63UPqr1f0u7+fnig7ANlyyPYG3LLUfhBT/d9VH0W644lqF8eZo0INEHfQf  +g4qvVVSTWfuC84ky5gTnWMbzB0iqVsZD3xw4wfSrSKyK6QFNESNdOo+1E0nz83I  cQAFD+zSSMLgodHCgA9GlGECAwEAAQ==  -----END PUBLIC KEY-----  **公钥2：**  -----BEGIN PUBLIC KEY-----  MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAymf92H5ljvvfTE8QjuOd  xv7YPOxXC05VceuSjtZN1aDb/4gqpWxDyMzRrPS8VRQTxkqWia4nd//zj+dheHNv  6+Emb3f00IyC2bcAFvDgQmnQB0sJZf2UI3mbMfLdnsIYW2YCbvxEiFYmUUOnh6xP  AnYFtZuvh9EDpyUwT95thQS23UEO2M1y5Q9SRUZo4EeQGb6/iqB6Q5FYabRqbsXe  Ckqxk1ENkPpuLkiQCtra++bICj4WbfVCOiiYpaN/faVud6qMHxsCxxkk+2p7kcs3  ZsCSEmLBzFNmzT32pMK9pq/rAXyXbGh4ECDuTdk1va/cCxIr5Ongven4oe4qGdnj  OCD9xPNfQZDSpYMaBcn1UveM9Rrv/GYaC9AgMnVvG5PaQOYKJzETU2gJm4rdPp/M  Hc9CvN30B6X9ewsLYIaA8ES/DRIqMG4GKAgMz0siROwLXMSkLXg1u4+mLeQzBQP5  TPJ5qwAwKJc6uPPoXo9ZmFnFW4THCoEJ+caax9M0Urg4+B6ids73C2u8A6xqVXld  ng0pAdt5exZqckhPWaWajFt4mmbUmlot7GU9PxV+NDhCn4YDmhBKQRin4lkuLilM  0/WmvnVxD7IhgXbDYrP7E1j/IO9VZQOGkntVT/BtvhJLQauF6J2bxyct9GD6Ahg4  BBKL1/FPLaDsmzWqbNiJKp8CAwEAAQ==  -----END PUBLIC KEY-----  **使用公钥1加密的对称密钥：**   |

首先对于RSA加密算法，公钥（n,e），私钥(n,d)，只知道n和e要解出d是极其困难的，除非我们可以算出phi(n) = (p-1)(q-1) ,由于给出的两个公钥的大整数有公因子，解析两个公钥得到以下两个大整数n1和n2。

n1:   
7638767497661761069070970754991369325425063094627901342381456628 6402944835767232938424078642982550665173547129319285780029207443 2517560147934296565646804636440319232118174807874707278193157951 0227911264325489542978726038418234717087300190407118433294118520 9334726187024436794795755376207947296628131312634365309918460422 9852078759250659887230164618217053184654632257026197939177514038 5470229063434675641567318889053455588258968854451372748800472793 9815801289901902372731318264452793945181148933872071830785078436 6603399048017626264140284123069450132380165349204708385658847188 1791229936014338236660522626712765500745070904660952707904352439 4404461378372079368412247299711953168007069944200448522849392328 4784588609625299335494520546908568332978745943998585734768939661 6013680541376983864185171844693119201036175257686472243323487637 3995362281588862110251166616513813337215732836528639314298764172 3981258561805886361917214660712608451187202188877282119338253380 4393611870918826195438353026254722467209249916204676742001858077 8291004642853256878397068523459360288842230549168637391568383002 0610512240092345516269641178633217279587340585381147938669982611 1803224461058972911979293271107892826886695320906347007700734229 88590964225250401

n2：

8257453230352727987205996083611924927467876120458769532622892111

6934226834632237529963438818338196764490373355991241721449614542

7027254956721988308512463763281120693615792222840534071393074980

7620772385067911263399091312082641479782008387047030703693284684

9547416412806596658317269602065036109002837409514774790093162333

1337880180057566279781633562179594032007854587452253456011800679

6087636993279776867893270096507914629702419972077312389484742620

6507768527392466652937756530318950877958047549901365112440395211

0758192421559847639791002451557155022419866908364191649354978851

2507363016531527867928341850025492895780887322523488147867126470

5624673739981845168881979994568378470217480791816735406033498184

8371709009902941634174705067621057569867546563489955228609184225

5584156427766703029135474988031623176554245662143644293663677923

8879645609520907530815703192211384732408279212827245608683817944

8027864861814855945170296255542403540294916048693047962499625508

9636804414005895693082341362778985436590428958128339493235920405

2773575657647600522999551515352635095297894182442717707889131728

4336320446076086808680961181640941049041256254665150119759613810

1582616965789228461197684403645635919638430639346958157331305976

65757383432415903

e:65537

计算两个n的最大公约数得

gcd：

26627466049142782101189461574085020996235089156991135799522067647594275206527359658599180991364006790412470178983959891166450396562210736651239193396713286992136142461550070059217212936987738288675942968812227629223275980884750849305664154083685317995658517078413156737417252300440129121261807308751775299709307288521049604030621405834040779873490914247498118977348086366949061671661649633292359659769411644172311163846383112345844023711735501538620108619130135669124180573103605584257477958903301414926536901348873911468112406031503884991837362914821473607550589146165095676280897853974555931846786448411416266568257

n/gcd：

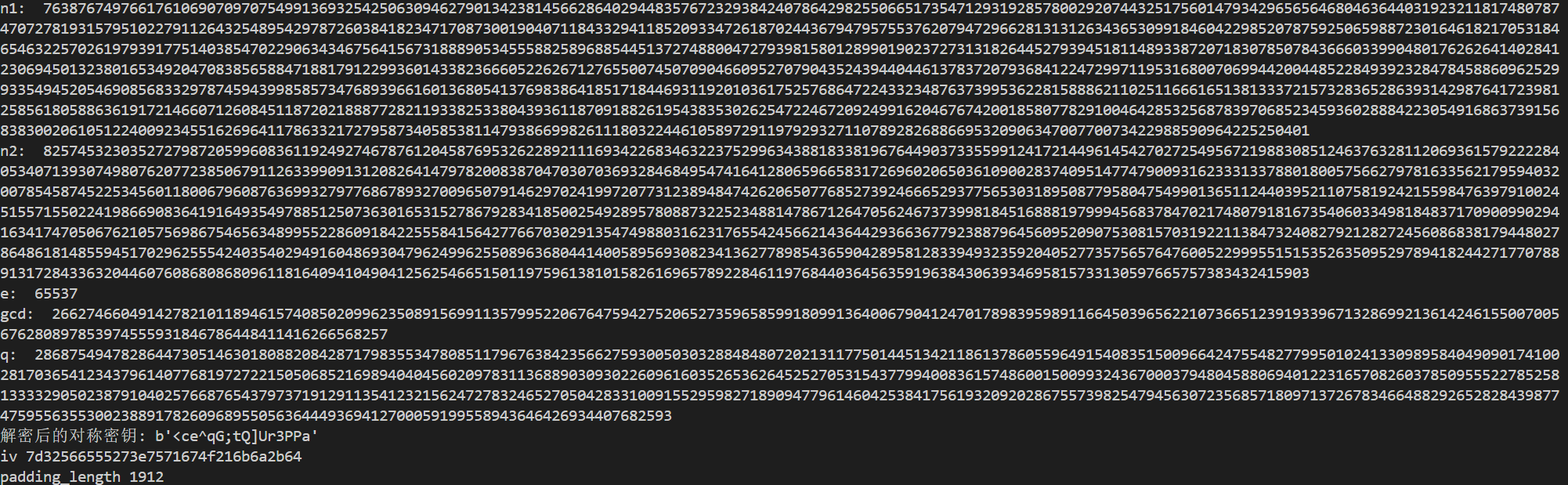
28687549478286447305146301808820842871798355347808511796763842356627593005030328848480720213117750144513421186137860559649154083515009664247554827799501024133098958404909017410028170365412343796140776819727221505068521698940404560209783113688903093022609616035265362645252705315437799400836157486001500993243670003794804588069401223165708260378509555227852581333329050238791040257668765437973719129113541232156247278324652705042833100915529598271890947796146042538417561932092028675573982547945630723568571809713726783466488292652828439877475955635530023889178260968955056364449369412700059199558943646426934407682593n

由于n的特点，猜测其为p、q，计算拼成私钥所需要的各个数值phi(n)和d，得到私钥后解密对称密钥得到：

b'<ce^qG;tQ]Ur3PPa'

接着对密文图像进行处理，先提取密文数据前16字节的IV，接着查看密文数据的最后一个像素，即后四个字节，得到填充像素为1912个，据此去除前16字节和后1912个像素得到真正的密文数据。根据密钥和IV创建AES解密器解密密文数据，对解密结果去除PKCS #7填充，得到真正的明文，将明文重构为图像保存在dec1.png中。

程序运行得到的各项数据：



解密后的结果：



RSA\_hack.py:

1. from Crypto.Cipher import AES

2. from PIL import Image

3. import numpy as np

4. from Crypto.Util.Padding import unpad

5. from Crypto.PublicKey import RSA

6. from Crypto.Util.number import inverse

7. from base64 import b64decode

8. from math import gcd

9. from cryptography.hazmat.primitives import serialization

10. from cryptography.hazmat.backends import default\_backend

11. from cryptography.hazmat.primitives.asymmetric.rsa import RSAPublicKey

12. from cryptography.hazmat.primitives.asymmetric import rsa

13. from cryptography.hazmat.primitives.asymmetric import padding

14. from cryptography.hazmat.primitives import hashes

15. import base64

16. from Crypto.Cipher import PKCS1\_OAEP

17. public\_key1\_text = """

18. -----BEGIN PUBLIC KEY-----

19. MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAuz20BUTcqVDjzEOKiJF9

20. 66LbQB/59lnXTj/SmiD07mV1XE03BLrWfi7jFh/iq5ZPzVXfbNPjHiojO9WRhWzr

21. wiQGZNVZ7qFoO/PzXOT8OyHyOMcrb6ogtCyFvDOeximr3M/ICmliU2JxbLSfteZj

22. AplHJVgs5bJ5LTW7eSy1x2Z5aOsHjesK3rkLi1yB2jM0MeaNIB/Enb82bBMKzAam

23. vN6tY8bQbEoRbTnlX6PUfkU9w7XsWLMa3QbpIH9mNam1Qz4ynCjWXcDo6KzYotUf

24. TgGlIIOOJKsAqgOgSHqTz83e8bBizPwJg+CxBzP4Ha8C9phc41i2GiEgDf4J1J0R

25. 0BZDcJEgZIlI+B5tlvJTy/uQyvmEP+hyMD8d83RdzLYy9h8u0MNHjJygY/Kktftp

26. wPtZPThpMOWWbOMM72a8Y2usz5rKTBAe+bN5QyELCErc/aQB0ABUSsNf4XxaQWbz

27. gJdb3hEvUkas0PfHui8UB6Yuaa7RmEE6EPIELx2WF2BGw1AG8vg5mi3I+HYxpk9W

28. mxy2gj63UPqr1f0u7+fnig7ANlyyPYG3LLUfhBT/d9VH0W644lqF8eZo0INEHfQf

29. +g4qvVVSTWfuC84ky5gTnWMbzB0iqVsZD3xw4wfSrSKyK6QFNESNdOo+1E0nz83I

30. cQAFD+zSSMLgodHCgA9GlGECAwEAAQ==

31. -----END PUBLIC KEY-----

32. """

33. public\_key2\_text = """

34. -----BEGIN PUBLIC KEY-----

35. MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAymf92H5ljvvfTE8QjuOd

36. xv7YPOxXC05VceuSjtZN1aDb/4gqpWxDyMzRrPS8VRQTxkqWia4nd//zj+dheHNv

37. 6+Emb3f00IyC2bcAFvDgQmnQB0sJZf2UI3mbMfLdnsIYW2YCbvxEiFYmUUOnh6xP

38. AnYFtZuvh9EDpyUwT95thQS23UEO2M1y5Q9SRUZo4EeQGb6/iqB6Q5FYabRqbsXe

39. Ckqxk1ENkPpuLkiQCtra++bICj4WbfVCOiiYpaN/faVud6qMHxsCxxkk+2p7kcs3

40. ZsCSEmLBzFNmzT32pMK9pq/rAXyXbGh4ECDuTdk1va/cCxIr5Ongven4oe4qGdnj

41. OCD9xPNfQZDSpYMaBcn1UveM9Rrv/GYaC9AgMnVvG5PaQOYKJzETU2gJm4rdPp/M

42. Hc9CvN30B6X9ewsLYIaA8ES/DRIqMG4GKAgMz0siROwLXMSkLXg1u4+mLeQzBQP5

43. TPJ5qwAwKJc6uPPoXo9ZmFnFW4THCoEJ+caax9M0Urg4+B6ids73C2u8A6xqVXld

44. ng0pAdt5exZqckhPWaWajFt4mmbUmlot7GU9PxV+NDhCn4YDmhBKQRin4lkuLilM

45. 0/WmvnVxD7IhgXbDYrP7E1j/IO9VZQOGkntVT/BtvhJLQauF6J2bxyct9GD6Ahg4

46. BBKL1/FPLaDsmzWqbNiJKp8CAwEAAQ==

47. -----END PUBLIC KEY-----

48. """

49.

50. encrypted\_key\_Base64 = """

51. 

52. """

53.

54. # 加载公钥

55. public\_key1 = serialization.load\_pem\_public\_key(public\_key1\_text.encode(), backend=default\_backend())

56. public\_key2 = serialization.load\_pem\_public\_key(public\_key2\_text.encode(), backend=default\_backend())

57.

58. # 获取n和e值

59. n1 = public\_key1.public\_numbers().n

60. n2 = public\_key2.public\_numbers().n

61. e = public\_key1.public\_numbers().e

62. print("n1: " , n1 ,"\nn2: ", n2 , "\ne: " , e)

63.

64. # 获取gcd

65. \_gcd = gcd(n1 , n2)

66. print("gcd: " , \_gcd)

67.

68. p = gcd(n1, \_gcd)  # p是n和公因子的最大公约数

69. q = n1 // p

70. print("p: " , p)

71. print("q: " , q)

72. phi = (p - 1) \* (q - 1)

73. d = inverse(e, phi)

74. #拼接私钥

75. private\_key = RSA.construct((n1 , e , d , p , q))

76.

77. encrypted\_symmetric\_key = base64.b64decode(encrypted\_key\_Base64)

78. cipher\_rsa = PKCS1\_OAEP.new(private\_key)

79. symmetric\_key\_base64 = cipher\_rsa.decrypt(encrypted\_symmetric\_key)

80. symmetric\_key = base64.b64decode(symmetric\_key\_base64)

81. print("解密后的对称密钥:", symmetric\_key)

82.

83. # 加载密文图像

84. image = Image.open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\enc1.png\\enc1.png")

85. image\_data = np.array(image)

86.

87. image\_bytes = image\_data.tobytes()

88.

89. # 从密文图像中提取 IV

90.

91. iv = image\_bytes[:16]

92. print("iv" , iv.hex())

93.

94. padding\_length = image\_bytes[-4:]

95. padding\_length = int(padding\_length.hex() , 16)

96. print("padding\_length" , padding\_length)

97.

98. encrypted\_data = image\_bytes[16 : ]

99. encrypted\_data = encrypted\_data[ : -4 \* padding\_length  ]

100.

101. # # 创建 AES 解密器

102. cipher = AES.new(symmetric\_key, AES.MODE\_CBC, iv)

103.

104. # # 解密密文图像数据

105. decrypted\_data\_pad = cipher.decrypt(encrypted\_data)

106. decrypted\_data\_unpad = unpad(decrypted\_data\_pad , AES.block\_size)

107.

108. dec\_image = Image.frombytes("RGBA" ,(1920 , 1080), decrypted\_data\_unpad)

109.

110. dec\_image.save("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB3\\enc1.png\\dec1.png")