密码学原理
实验三：公钥加密

学号：2022112266　　　　姓名：魏圣卓

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。

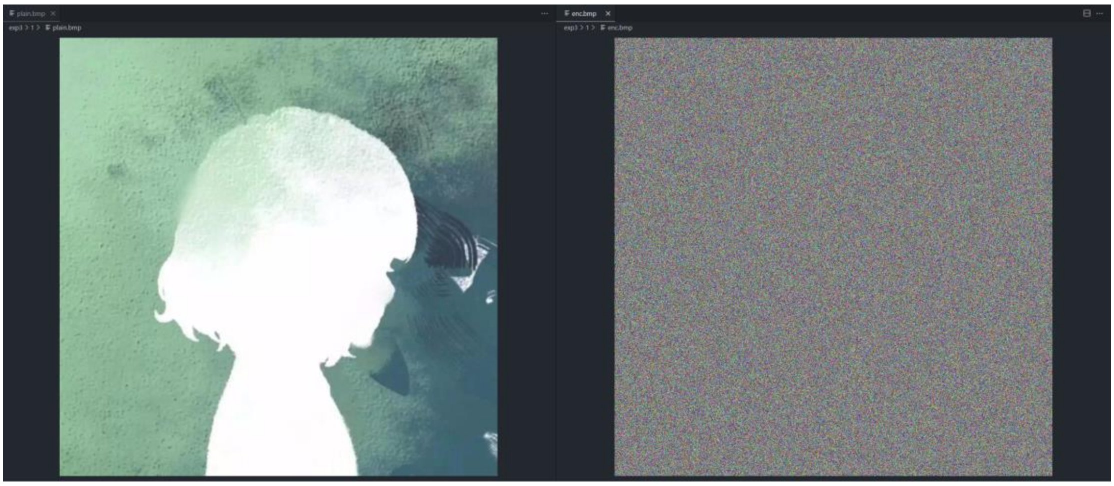
1、 公钥加密

(1) 使用密码学工具实现混合加密过程

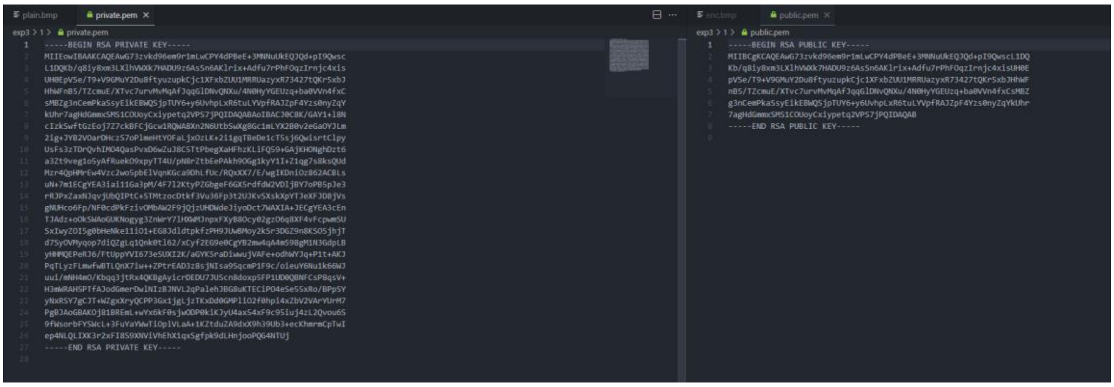
要求：生成 RSA 密钥对（公钥 1 和私钥 1），用公钥 1 加密对称密钥并采用对称加密方法加密图片，用私钥 1 解密对称密钥，然后解密图片。（你可以使用实验二中的对称加密方案）

基本原理同实验 2，首先随机生成 AES 的密钥和 iv 值，然后使用其对图片进行加密

加密前后图片对比：



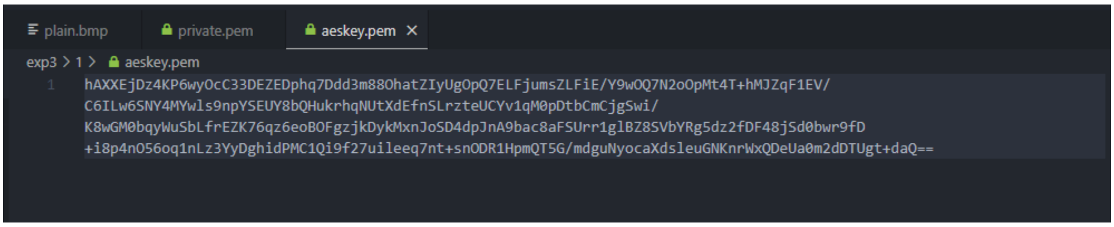
随机生成一对 RSA 密钥，并存于本地.pem 文件内



```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAwG73zvkd96em9r1mLwCPY4dPBeE+3MNNuUkEQJQd+pI9Qwsc
L1DQKb/q8iy8xm3LXlhVWXk7HADU9z6AsSn6AKlrix+Adfu7rPhFOqzIrnjc4xis
UH0EpVSe/T9+V9GMuY2Du8ftyuzupkCjc1XFxbZUU1MRRUazyxR73427tQKr5xbJ
HhWFnB5/TZcmuE/XTvc7urvMvMqAfJqqGlDNvQNXu/4N0HyYGEUzq+ba0VVn4fxC
sMBZg3nCemPkaSsyEikEBWQSjpTUY6+y6UvhpLxR6tuLYVpfRAJZpF4Yzs0nyZqY
kUhr7agHdGmmxSMS1COUoyCx1ypetq2VPS7jPQIDAQABAoIBACJ0C8K/GAY1+l8N
cIzkSwftGzEoj7Z7ckBFCjGcw1RQWA8Xn2N6UtbSwXg8Gc1mLYX2B0v2eGaOYJLm
2ig+JYB2VOarOHczS7oPlmeHtYOFaLjxOzLK+2i1gqTBeDe1cT5sj6QwisrtClpy
UsFs3zTDrQvhIMO4QasPvxD6wZuJ8C5TtPbegXaHFhzKLiFQS9+GAjKHONghDzt6
a3Zt9veg1o5yAfRuekO9xpyTT4U/pNBrZtbEePAkh9OGg1kyY1I+Z1qg7s8ksQUd
Mzr4QpHMrEw4Vzc2woSpbElVqnKGca9DhLfUc/RQxXX7/E/wgIKDniOz862ACBLs
uN+7m1ECgYEA3iai11Ga3pM/4F7l2KtyPZGbgeF6GX5rdfdW2VDljBY7oPB5pJe3
rRJPxZaxNJqvjUbQIPtC+STMtzocDtkf3Vu36Fp3t2UJKvSXskXpYTJeXFJD8jVs
gNUHco6Fp/NF0cdPkFzivOMbAW2F9jQjzUHDWdeJiyoDct7WAXIA+JECgYEA3cEn
TJAdz+oOkSWAoGUKNogyg3ZnWrY7lHXWMJnpxFXyB8Ocy02gzO6q8XF4vFcpwm5U
5xIwyZOI5g0bHeNke11iO1+EG8JdldtpkfzPH9JUwBMoy2kSr3DGZ9n8KSO5jhjT
d7SyOVMyqop7d1QZgLq1Qnk0tl62/xCyf2EG9e0CgYB2mw4qA4m598gM1N3GdpLB
yHHMQEPeRJ6/FtUppYVI673e5UXI2K/aGYK5raD1wwujVAFe+odhWYJq+P1t+AKJ
PqTLyzFLmwfwBTLQnX7iw++ZPtrEAD3z8sjNIsa9SqcmP1F9c/oieuY6Nu1k66WJ
uui/mNH4mO/Kbqq3jtRx4QKBgAy1crDEDU7JUScn8doxp5FP1UD0QBNFCsP8qsV+
H3mWRAHSPTfAJodGmerDwlNIzBJNVL2qPalehJBG8uKTECiPO4eSe5SxRo/BPpSY
yNxRSY7gCJT+WZgxXryQCPP3Gx1jgLjzTKxDd0GMPliO2f0hpi4xZbV2VArYUrM7
PgBJAoGBAKOj81BREmL+wYx6kF0sjwODP0kiKJyU4ax54xF9c95iuj4zL2Qvou6S
9fWsorbFYSWcL+3FuYaYWwTiOpiVLaA+1KZtduZA9dxX9h39Ub3+ecKhmrmCpTwI
ep4NLQLIXK3r2xFI8S9XNViVhEhX1qxSgfpk9dLHnjooPQG4NTUj
-----END RSA PRIVATE KEY-----
```

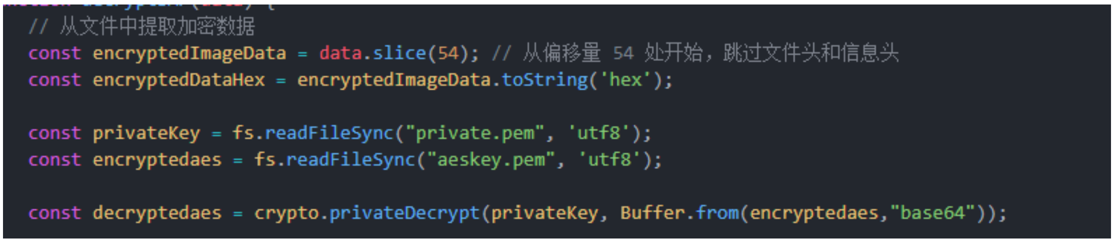
```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAwG73zvkd96em9r1mLwCPY4dPBeE+3MNNuUkEQJQd+pI9QwscL1DQ
Kb/q8iy8xm3LXlhVWXk7HADU9z6AsSn6AKlrix+Adfu7rPhFOqzIrnjc4xisUH0E
pVSe/T9+V9GMuY2Du8ftyuzupkCjc1XFxbZUU1MRRUazyxR73427tQKr5xbJHhWF
nB5/TZcmuE/XTvc7urvMvMqAfJqqGlDNvQNXu/4N0HyYGEUzq+ba0VVn4fxCsMBZ
g3nCemPkaSsyEikEBWQSjpTUY6+y6UvhpLxR6tuLYVpfRAJZpF4Yzs0nyZqYkUhr
7agHdGmmxSMS1COUoyCx1ypetq2VPS7jPQIDAQAB
-----END RSA PUBLIC KEY-----
```

使用此 RSA 密钥对 json 化的 AES 密钥和 iv 值进行加密，并将其保存到本地 aeskey.pem 中



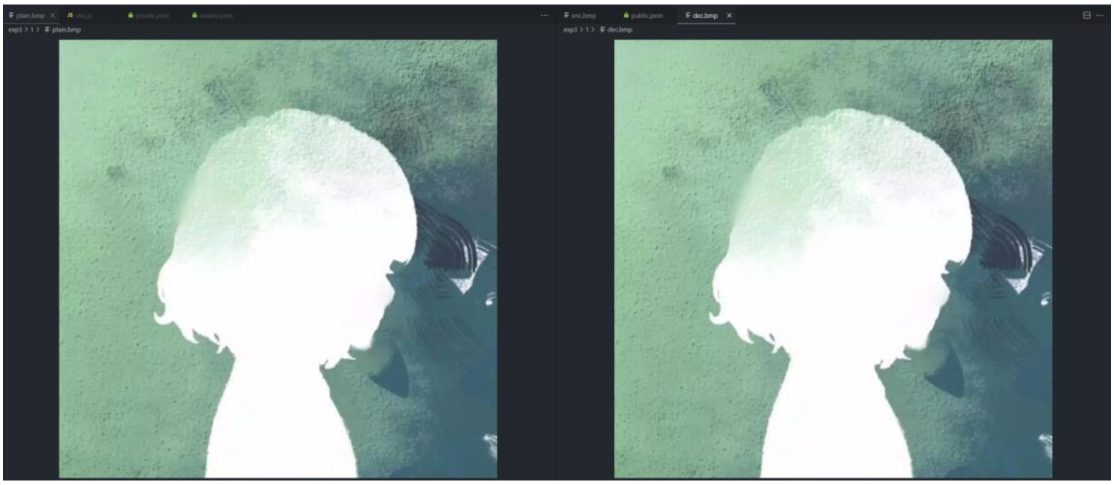
```
hAXXEjDz4KP6wyOcC33DEZEDphq7Ddd3m88OhatZIyUgOpQ7ELFjumsZLFiE/Y9wOQ7N2oOpMt4T+hMJZqF1EV/
C6ILw6SNY4MYwls9npYSEUY8bQHukrhqNUtXdEfnSLrzteUCYv1qM0pDtbCmCjgSwi/
K8wGM0bqyWuSbLfrEZK76qz6eoBOFgzjkDykMxnJoSD4dpJnA9bac8aFSUrr1glBZ8SVbYRg5dz2fDF48jSd0bwr9fD
+i8p4nO56oq1nLz3YyDghidPMC1Qi9f27uileeq7nt+snODR1HpmQT5G/mdguNyocaXdsleuGNKnrWxQDeUa0m2dDTUgt+daQ==
```

解密时，首先读取本地加密后的 aeskey，然后使用私钥对其进行解密



```
// 从文件中提取加密数据
const encryptedImageData = data.slice(54); // 从偏移量 54 处开始，跳过文件头和信息头
const encryptedDataHex = encryptedImageData.toString('hex');

const privateKey = fs.readFileSync("private.pem", 'utf8');
const encryptedaes = fs.readFileSync("aeskey.pem", 'utf8');

const decryptedaes = crypto.privateDecrypt(privateKey, Buffer.from(encryptedaes,"base64"));
```

然后再使用 AES 密钥和 iv 值对图片进行解密



成功解密出图片

2、 相同因子公钥分析

(1) 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 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 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
```

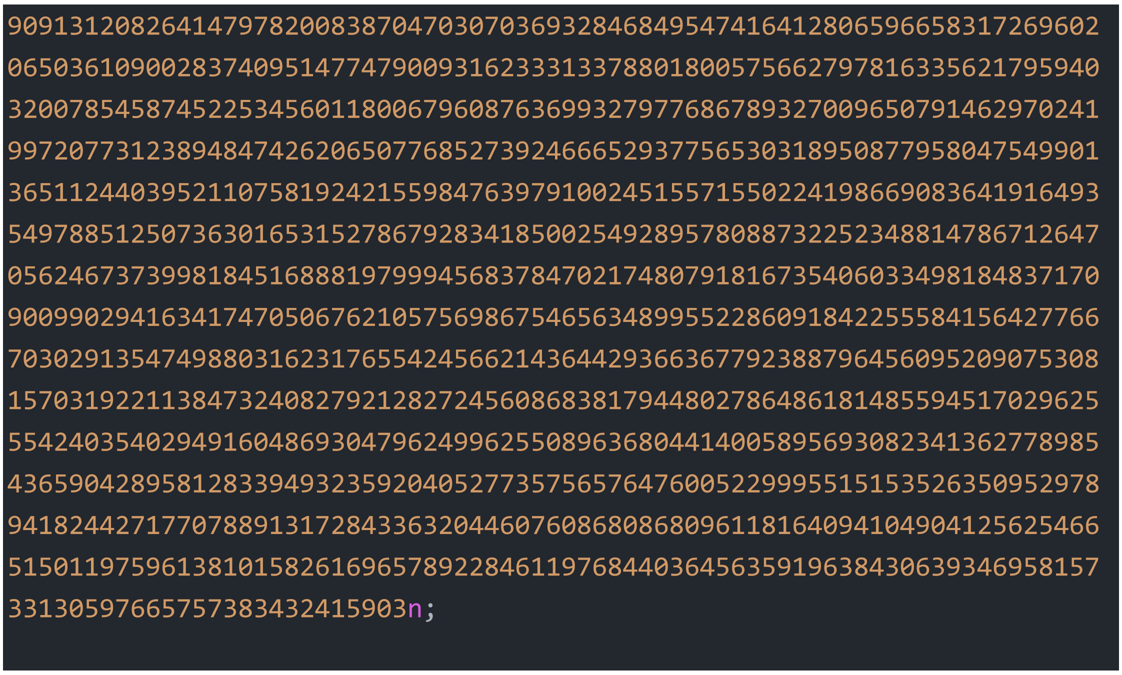
```
AnYFtZuvh9EDpyUwT95thQS23UEO2M1y5Q9SRUZo4EeQGb6/iqB6Q5FYabRqbsXe
Ckqxk1ENkPpuLkiQCtra++bICj4WbfVCOiiYpaN/faVud6qMHxsCxxkk+2p7kcs3
ZsCSEmLBzFNmzT32pMK9pq/rAXyXbGh4ECDuTdk1va/cCxIr5Ongven4oe4qGdnj
OCD9xPNfQZDSpYMaBcn1UveM9Rrv/GYaC9AgMnVvG5PaQOYKJzETU2gJm4rdPp/M
Hc9CvN30B6X9ewsLYIaA8ES/DRIqMG4GKAgMz0siROwLXMSkLXg1u4+mLeQzBQP5
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：

```
MzhKNQx+U8ltsj5is29pSwu7yqdgoWPWIhgEwUTz3ywE84ue99Z7T/AISGOuyud6ET4E8xXFS/7wadzwY
j3yL6dQrw+F9KFPJRNkTDQll0Re+3kkGt2+M68HJRvmIcJaD1/0PNTv9gek5PdL59TNq/VerwqXusAIIO
dclwhb+U1EGJzJ0RS+8Wyp/+PU4J5P2mtFSak5SKNzDB8yg00uyhRBZGriQzw+QQRZanWJYs45UFYIP+9
ZMUK3lOkf3b8CT+qGW/HcDFwG59hn59PUvN8UFER3PcOTIRD/+RBSKoi1Sdr7uxvQ3XTBvFJKlDMp1es4
yzewmOgluBY2DtGV+aAbLzu5Sy6EfF7tJgid8V9T9ZQ8nqW9vtWkt6Y2okRhdkpX+E+y240gU1BEHOUNg
lM6oJ1b0nGiAL5cjUtX0IknEAsZR/U2ztsMQRzvy10xJpIgipKB52aNh6BnYzFH4DYndfehKh1NjVckcJ
OK+krTiUNwQMNhRYSZ8v1pZH6jR96TuDPib1KcJopjaGdf9zNa2bkdJ7NSWTe9j1jHMPJYjrP6XCefsix
RTWp5dEz3KgzWEgGBHmIhz2SYYWLcy0SKb3ljYFUrY6tDwVRC+Srkk4GOeS09OvxT3r9E/JdaiA9BXuRj
rV7LeCAW18AwbpZEaTHxjrVcoZ5sWpNasCI=
```

直接解析两个公钥，得到两个 n 值

```
const bigInt1 =
76387674976617610690709707549913693254250630946279013423814566286402944
83576723293842407864298255066517354712931928578002920744325175601479342
96565646804636440319232118174807874707278193157951022791126432548954297
87260384182347170873001904071184332941185209334726187024436794795755376
20794729662813131263436530991846042298520787592506598872301646182170531
84654632257026197939177514038547022906343467564156731888905345558825896
88544513727488004727939815801289901902372731318264452793945181148933872
07183078507843666033990480176262641402841230694501323801653492047083856
58847188179122993601433823666052262671276550074507090466095270790435243
94404461378372079368412247299711953168007069944200448522849392328478458
86096252993354945205469085683329787459439985857347689396616013680541376
98386418517184469311920103617525768647224332348763739953622815888621102
51166616513813337215732836528639314298764172398125856180588636191721466
07126084511872021888772821193382533804393611870918826195438353026254722
46720924991620467674200185807782910046428532568783970685234593602888422
30549168637391568383002061051224009234551626964117863321727958734058538
11479386699826111803224461058972911979293271107892826886695320906347007
7007342298859096422525040ln;
const bigInt2 =
82574532303527279872059960836119249274678761204587695326228921116934226
83463223752996343881833819676449037335599124172144961454270272549567219
88308512463763281120693615792222840534071393074980762077238506791126339
```



```
909131208264147978200838704703070369328468495474164128065966583172696O2
065036109002837409514774790093162333133788018005756627978163356217959 40
320078545874522534560118006796087636993279776867893270096507914629702 41
997207731238948474262065077685273924666529377565303189508779580475499 01
365112440395211075819242155984763979100245155715502241986690836419164 93
549788512507363016531527867928341850025492895780887322523488147867126 47
056246737399818451688819799945683784702174807918167354060334981848371 70
900990294163417470506762105756986754656348995522860918422555841564277 66
703029135474988031623176554245662143644293663677923887964560952090753 08
157031922113847324082792128272456086838179448027864861814855945170296 25
554240354029491604869304796249962550896368044140058956930823413627789 85
436590428958128339493235920405277357565764760052299955151535263509529 78
941824427177078891317284336320446076086808680961181640941049041256254 66
515011975961381015826169657892284611976844036456359196384306393469581 57
33130597665757383432415903n;
```

e 值为固定的 65573

计算两者的最大公约数，得到：

```
266274660491427821011894615740850209962350891569911357995220676475942 75

206527359658599180991364006790412470178983959891166450396562210736651 23

919339671328699213614246155007005921721293698773828867594296881222762 92

232759808847508493056641540836853179956585170784131567374172523004401 29

121261807308751775299709307288521049604030621405834040779873490914247 49

811897734808636694906167166164963329235965976941164417231116384638311 23

458440237117355015386201086191301356691241805731036055842574779589033 01

414926536901348873911468112406031503884991837362914821473607550589146 16

5095676280897853974555931846786448411416266568257
```

计算 n/gcd:

```
286875494782864473051463018088208428717983553478085117967638423566275 93

005030328848480720213117750144513421186137860559649154083515009664247 55

482779950102413309895840490901741002817036541234379614077681972722150 50

685216989404045602097831136889030930226096160352653626452527053154377 99
```

```
40083615748600150099324367000379480458806940122316570826037850955522785
25813333290502387910402576687654379737191291135412321562472783246527050
42833100915529598271890947796146042538417561932092028675573982547945630
72356857180971372678346648829265282843987747595563553002388917826096895
50563644493694127000591995589436464269344076825 93n
```

猜测认为此为公钥 1 的 p、q 值

计算私钥所需相关数值

```
RSA Parameters:
n:
bb3db40544dca950e3cc438a88917deba2db401ff9f659d74e3fd29a20f4ee65755c4d3704bad67e2ee
3161fe2ab964fcd55df6cd3e31e2a233bd591856cebc2240664d559eea1683bf3f35ce4fc3b21f238c7
2b6faa20b42c85bc339ec629abdccfc80a69625362716cb49fb5e66302994725582ce5b2792d35bb792
cb5c7667968eb078deb0adeb90b8b5c81da333431e68d201fc49dbf366c130acc06a6bcdead63c6d06c
4a116d39e55fa3d47e453dc3b5ec58b31add06e9207f6635a9b5433e329c28d65dc0e8e8acd8a2d51f4
e01a520838e24ab00aa03a0487a93cfcddef1b062ccfc0983e0b10733f81daf02f6985ce358b61a2120
0dfe09d49d11d01643709120648948f81e6d96f253cbfb90caf9843fe872303f1df3745dccb632f61f2
ed0c3478c9ca063f2a4b5fb69c0fb593d386930e5966ce30cef66bc636baccf9aca4c101ef9b3794321
0b084adcfda401d000544ac35fe17c5a4166f380975bde112f5246acd0f7c7ba2f1407a62e69aed1984
13a10f2042f1d96176046c35006f2f8399a2dc8f87631a64f569b1cb6823eb750faabd5fd2eefe7e78a
0ec0365cb23d81b72cb51f8414ff77d547d16eb8e25a85f1e668d083441df41ffa0e2abd55524d67ee0
bce24cb98139d631bcc1d22a95b190f7c70e307d2ad22b22ba40534448d74ea3ed44d27cfcdc8710005
0fecd248c2e0a1d1c2800f469461
e: 10001
d:
8275e09b973a6462af05e0bd82a054eb3cc2a530627aaa6860e6093848e43fc2c37df772f141b9379c3
5af71b1feba318a315e0636b0559128918521eaa454b42563ab18189c332c2c31b28c342426936570f8
1d24a59639d397aee50ca8a7da4e751bbd5d661c1148546499af2502318a58cf055beb036a78fcaa4e2
8bfd35a6c179542e338348fb0c1016e082fdf636bad67014e0131a44c280b3087adceb96431cd86762c
```

```
2d836ce65d1241554a51b65249cc8e94e48fc243e6d7bd808069829bb0a9abae890e6379d30cf28faa4

b30cc3c164d90f0f0fa84a2463dda057d3cec8b689bc8b45c85cef2e4dbe8dacb26c44c5c3fb04a0e70

862f4dec2d307de9e6b6c7a71e5e19f283217c35be2632cb4f5178999acff6a32b4060a28730083d723

08d5979f05fe02c2cbc38df733d86ff7f0df0b02adb3a9d3675aca749d00695c542122edc4688ec456b

28a63df337626c89bbe54aca8bc39c0f1ad3d80a66f36f9c288b0eb031a9e84a6bff9a0e6ed876974a1

850524cf805f95523a793802023cbec18ac14487f8b227641ced6455a99812670ba840d70b3b35b0c9b

e810f1f99703c087ceaa02bd87c02eaa1abe5f573501a20b2edc266bbe1e1af709cccfe62473a1715c5

eba9b886f2311a20a52c773c89890982b410a65725dd54c9591d6b2d4f72059d2332c9ce29560f769cf

fc630d273b97984ac4a0cd613801

p:

d2ee1f0fc20678b9c667e445bd93fcd04cff5b85cb9e7b2a6b4aed8d28909732cbf2951b094d307e4cd

5d979dfc0a6d3dc507a3ca1997130f26fc2ea72483b1e767cb2a396f4669647bd9f9b205b82872276c7

06a178fce7c35effbd0f7a1e729258317d8d96120546384137379f6ac7dc52ab3b0f98bcc99b24ae1e2

dcf57aac1cb5e41084399673683799abc422077ba23cb3adfb44e4b07d5e4b228b0cc9f875ad62cc365

46bde985af92fea41ef76705129ed206d8ec86c8768b5ee7b4571aab2d20772dad68e91ffe398ee2e07

40552ee16c56323a4e932b4856cd7c0c95b35e30d7eb61271e23578c7401692f720b726366b748dcdf6

6080294c947241

q:

e33fc952272c64482b972cc165d8dd351cbae961d795d099afddbead06c27be0ad5907c0b13546f4ac9

8087df3c8396bc137ebe55112f54b42bb897763cceb508bb403a6a3f86199ccfc81d96d77a8dbedbf29

6adce5a923ef5bcc7b9bbe7f3b784664351621ba8ade9413b2d36deb261c18a77a62eb396c1bffe5b95

11f32f27d7f69733e28e55c7567f9fe6777e8ec1f6ed300c5322e9febcfe76b37208004f618fbf6460c

4ba8f909a0b116225998532b25b5aabbda9cbe68524dcd32c5b3f06652f4ae9970f1b4d4acf565e4a69

79e633df4bdeab0002f7a511334375efde7e102a83c322a5fb984449ed1713bd44383b892bb3f318d9a

e56ee0afb95a21

d % (p-1):

2ef60306601d4ebd95b5c5b5dd3a587c0bd88288ec4791866532de66a21467055130e60f89e79c97520

5fe904ac7ba9cca9d4449c174c725ada442522e3574ac189734b285464e9cc0c63005ba9385210f0440

d57217dcbec0d9a2b875fcab50b8d257da6b2e713ea95e96ee733cd3ea415c4a7130ccc1a3651e2a81d

f931d229058f5d826bdde2683958bfa6e558ea2f95744c50bcf39eb2c886b8d5acfda0ede74d3e04d74
```

```
7be6865a5437b6e7d559219f9880b1a402a1ce744d28446df3d1bcc74599c7ceec9fc1df749b5dfb971
b5da014100fcc3dc056870a77f63ea6d66daf36a8e78023e52506e0a803d5c2ccb084f646017086adc7
5fbec8f31acdc1
```

d % (q-1):

```
2036888936b85a36c8c713370f02e0c8889c0e3d2a5c3847b0e054091eee2eb5389d6835ee68c8f8220
7231d0793986027f18f64e04b39ae43756615207689dccf10e37b99beb9c71d81a5be768001d5bb2bdf
e9200b0620ebeb2d5f50427eb2a5680508696cba8eb8e270b066bc2c1b28dc6dd131787a995296c1afb
54a20e4b2d97ecf313e86d4c667c8ce41488422048aa72a72c99c465b1d0b86ed37942e1fdc86caf03f
09f297eabf769ad69fb0b20969bd08c8bc427fd335e78570ef0984c85c7606d2056a9de18feaf158aa2
ccfeced7965fa809c661e86a39c41451be4e4c685a692cde2e7df5202d9599dcb870e82f979c58976b3
4056a94eff8d41
```

q^{-1} mod p:

```
56189c322b4f1d9ab48b1d6f6748eed3896cc718dda6ae18fa7ecac3758fe1ca0c3ae64e65d6f3bc602
4fb88ac37fb214772439e16dbc25890117c0b65db300b6d91de8a9a39565258f8a2cc80654bc5780ada
ad1741c712b37d2f51df1870d1b52122bef7ceb22479520dc0e76e52e9a3908e0cbdc31268f130e3741
b3b63b12fdf6a6e6eef7126270a474fb09504b45eaf199d47ff34157689afdf177dca51c4a29b1dcd5c
3585b331099d7a8a84b9632260943f20dda78d9ca091b5d157a92becfdc5f4c9ab93e68ddb7077c1588
7e4cff9c71a90af80737723068be7afb3b9f2b4d17c921409d5aa1acf2ee3c0a02de6bcce9ab57c4096
a65178eb801112
```

拼接后可得到私钥

```
-----BEGIN PRIVATE KEY-----
MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQC7PbQFRNypUOPMQ4qIkX3rottAH/n2Wdd
OP9KaIPTuZXVcTTcEutZ+LuMWH+Krlk/NVd9s0+MeKiM71ZGFbOvCJAZk1VnuoWg78/Nc5Pw7IfI4xytvqi
C0LIW8M57GKavcz8gKaWJTYnFstJ+15mMCmUclWCzlsnktNbt5LLXHZnlo6weN6wreuQuLXIHaMzQx5o0gH
8SdvzZsEwrMBqa83q1jxtBsShFtOeVfo9R+RT3DtexYsxrdBukgf2Y1qbVDPjKcKNZdwOjorNii1R9OAaUg
g44kqwCqA6BIepPPzd7xsGLM/AmD4LEHM/gdrwL2mFzjWLYaISAN/gnUnRHQFkNwkSBkiUj4Hm2W8lPL+5D
K+YQ/6HIwPx3zdF3MtjL2Hy7Qw0eMnKBj8qS1+2nA+1k9OGkw5ZZs4wzvZrxja6zPmspMEB75s3lDIQsISt
z9pAHQAFRKw1/hfFpBZvOAl1veES9SRqzQ98e6LxQHpi5prtGYQToQ8gQvHZYXYEbDUAby+DmaLcj4djGmT
1abHLaCPrdQ+qvV/S7v5+eKDsA2XLI9gbcstR+EFP931UfRbrjiWoXx5mjQg0Qd9B/6Diq9VVJNZ+4LziTL
```

mBOdYxvMHSKpWxkPfHDjB9KtIrIrpAU0RI106j7UTSfPzchxAAUP7NJIwuCh0cKAD0aUYQIDAQABAoICAIJ
14JuXOmRirwXgvYKgVOs8wqUwYnqqaGDmCThI5D/Cw333cvFBuTecNa9xsf66MYoxXgY2sFWRKJGFIeqkVL
QlY6sYGJwzLCwxsow0JCaTZXD4HSSlljnTl67lDKin2k51G71dZhwRSFRkma8lAjGKWM8FW+sDanj8qk4ov
9NabBeVQuM4NI+wwQFuCC/fY2utZwFOATGkTCgLMIetzrlkMc2Gdiwtg2zmXRJBVUpRtlJJzI6U5I/CQ+bX
vYCAaYKbsKmrrokOY3nTDPKPqkswzDwWTZDw8PqEokY92gV9POyLaJvItFyFzvLk2+jayybETFw/sEoOcIY
vTewtMH3p5rbHpx5eGfKDIXw1viYyy09ReJmaz/ajK0BgoocwCD1yMI1ZefBf4CwsvDjfcz2G/38N8LAq2z
qdNnWsp0nQBpXFQhIu3EaI7EVrKKY98zdibIm75UrKi8OcDxrT2Apm82+cKIsOsDGp6Epr/5oObth2l0oYU
FJM+AX5VSOnk4AgI8vsGKwUSH+LInZBztZFWpmBJnC6hA1ws7NbDJvoEPH5lwPAh86qAr2HwC6qGr5fVzUB
ogsu3CZrvh4a9wnMz+Ykc6FxXF66m4hvIxGiClLHc8iYkJgrQQplcl3VTJWR1rLU9yBZ0jMsnOKVYPdpz/x
jDSc7l5hKxKDNYTgBAoIBAQDS7h8PwgZ4ucZn5EW9k/zQTP9bhcueeyprSu2NKJCXMsvylRsJTTB+TNXZed
/AptPcUHo8oZlxMPJvwupySDsednyyo5b0ZpZHvZ+bIFuChyJ2xwahePznw17/vQ96HnKSWDF9jZYSBUY4Q
Tc3n2rH3FKrOw+YvMmbJK4eLc9XqsHLXkEIQ5lnNoN5mrxCIHe6I8s637ROSwfV5LIosMyfh1rWLMNlRr3p
ha+S/qQe92cFEp7SBtjshsh2i17ntFcaqy0gdy2taOkf/jmO4uB0BVLuFsVjI6TpMrSFbNfAyVs14w1+thJ
x4jV4x0AWkvcgtyY2a3SNzfZggClMlHJBAoIBAQDjP8lSJyxkSCuXLMFl2N01HLrpYdeV0Jmv3b6tBsJ74K
1ZB8CxNUb0rJgIffPIOWvBN+vlURL1S0K7iXdjzOtQi7QDpqP4YZnM/IHZbXeo2+2/KWrc5akj71vMe5u+f
zt4RmQ1FiG6it6UE7LTbesmHBinemLrOWwb/+W5UR8y8n1/aXM+KOVcdWf5/md36OwfbtMAxTIun+vP52s3
IIAE9hj79kYMS6j5CaCxFiJZmFMrJbWqu9qcvmhSTc0yxbPwZlL0rplw8bTUrPVl5KaXnmM99L3qsAAvelE
TNDde/efhAqg8MipfuYREntFxO9RDg7iSuz8xjZrlbuCvuVohAoIBAC72AwZgHU69lbXFtd06WHwL2IKI7E
eRhmUy3maiFGcFUTDmD4nnnJdSBf6QSse6nMqdREnBdMclraRCUi41dKwYlzSyhUZOnMDGMAW6k4UhDwRA1
XIX3L7A2aK4dfyrULjSV9prLnE+qV6W7nM80+pBXEpxMMzBo2UeKoHfkx0ikFj12Ca93iaDlYv6blWOovlX
RMULzznrLIhrjVrP2g7edNPgTXR75oZaVDe259VZIZ+YgLGkAqHOdE0oRG3z0bzHRZnHzuyfwd90m137lxt
doBQQD8w9wFaHCnf2PqbWba82qOeAI+UlBuCoA9XCzLCE9kYBcIatx1++yPMazcECggEBACA2iIk2uFo2yM
cTNw8C4MiInA49Klw4R7DgVAke7i61OJ1oNe5oyPgiByMdB5OYYCfxj2TgSzmuQ3VmFSB2idzPEON7mb65x
x2Bpb52gAHVuyvf6SALBiDr6y1fUEJ+sqVoBQhpbLqOuOJwsGa8LBso3G3RMXh6mVKWwa+1SiDkstl+zzE+
htTGZ8jOQUiEIgSKpypyyZxGWx0Lhu03lC4f3IbK8D8J8pfqv3aa1p+wsglpvQjIvEJ/0zXnhXDvCYTIXHY
G0gVqneGP6vFYqizP7O15ZfqAnGYehqOcQUUb5OTGhaaSzeLn31IC2Vmdy4cOgvl5xYl2s0BWqU7/jUECgg
EBAFYYnDIrTx2atIsdb2dI7tOJbMcY3aauGPp+ysN1j+HKDDrmTmXW87xgJPuIrDf7IUdyQ54W28JYkBF8C
2XbMAttkd6KmjlWUlj4osyAZUvFeArarRdBxxKzfS9R3xhw0bUhIr73zrIkeVINwOduUumjkI4MvcMSaPEw
43QbO2OxL99qbm7vcSYnCkdPsJUEtF6vGZ1H/zQVdomv3xd9ylHEopsdzVw1hbMxCZ16ioS5YyJglD8g3ae
NnKCRtdFXqSvs/cX0yauT5o3bcHfBWIfkz/nHGpCvgHN3IwaL56+zufK00XySFAnVqhrPLuPAoC3mvM6atX

```
xAlqZReOuAERI=

-----END PRIVATE KEY-----
```

解密，得到

```
Decrypted Text:
PGNlXnFHO3RRXVVyM1BQYQ==
```

Base64 解密得到 AES 密钥：

```
3c63655e71473b74515d557233505061
```

分析密文图片，为标准 png 格式，可使用 pngjs 包处理

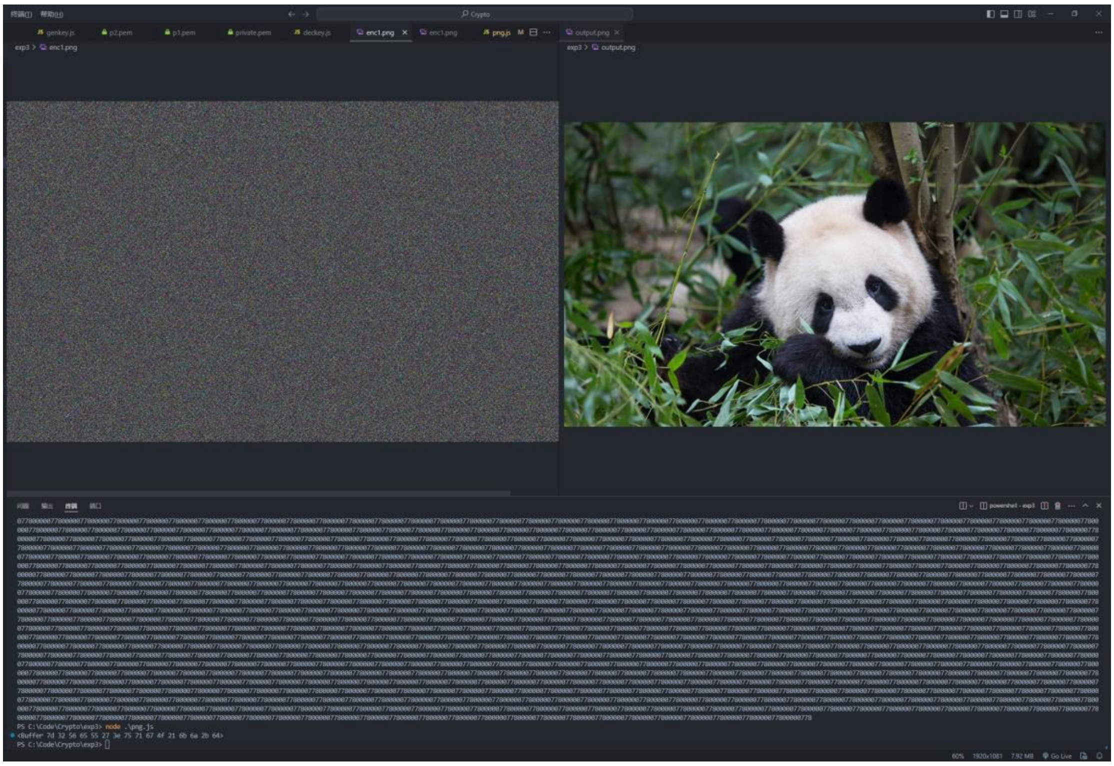
查看密文图片，大小为 1920*1081，由拼接方式可知原图片大小为 1920*1080

解析密文图片的最后一个像素，转译值为 0x00000778，即填充像素为 1912 个像素，对应密文前 128 占用的 1920-1912=8 个像素

读取密文前 8 像素，转译得到 128 位的 iv 值：7d 32 56 65 55 27 3e 75 71 67 4f 21 6b 6a 2b 64

将密文去除前 8 位和后 1912 像素后进行 AES 解密，使用前文破解的密钥和 iv 值。

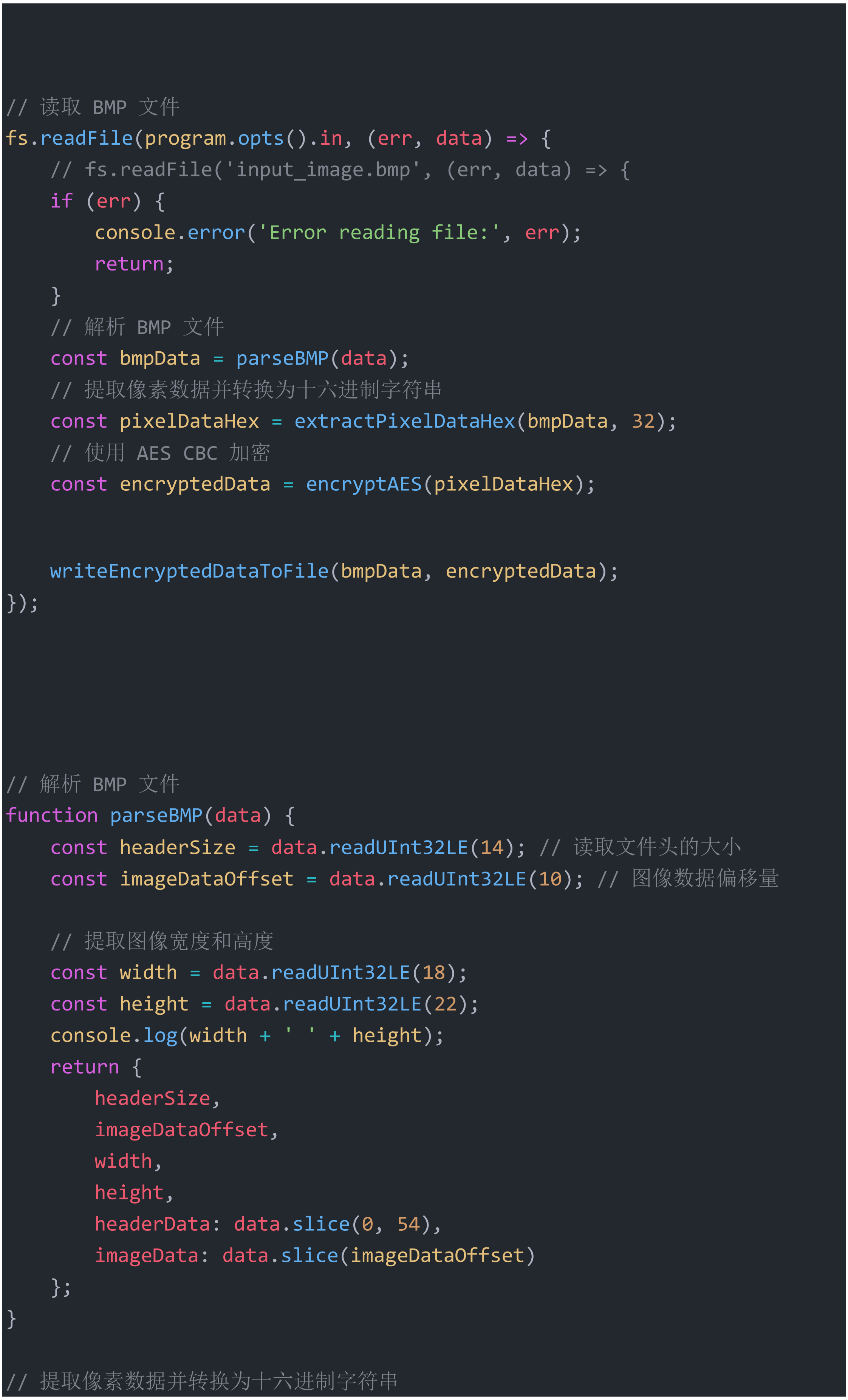
得到明文后，其长度对应 1920*1080 的图片，将其生成新的明文 png 图片



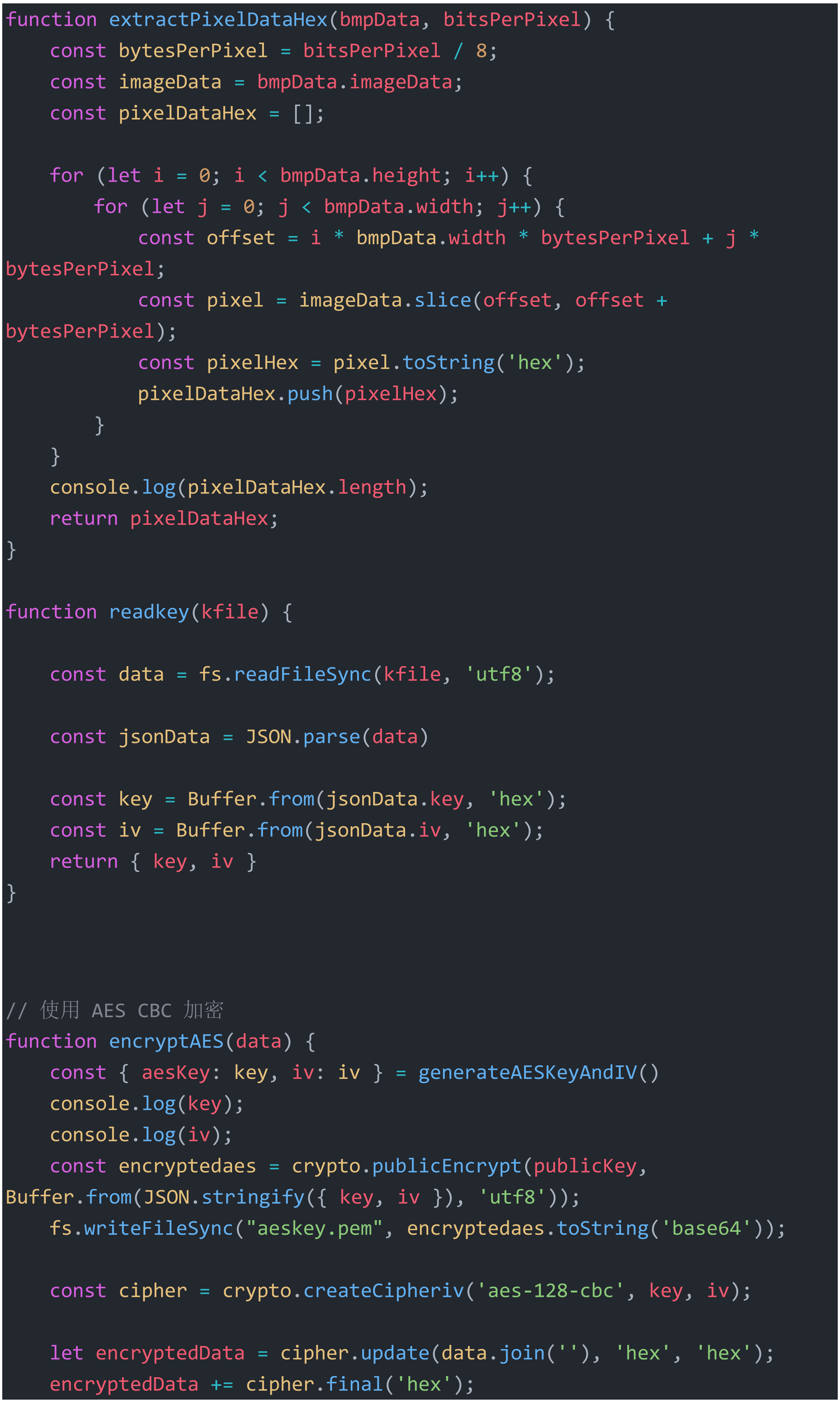
```
PS C:\Code\Crypto\exp3> node .\png.js
<Buffer 7d 32 56 65 55 27 3e 75 71 67 4f 21 6b 6a 2b 64>
PS C:\Code\Crypto\exp3>
```

Enc.js

```
const fs = require('fs');
const crypto = require('crypto');
const { program } = require('commander');

program
    .option('-k,--key <key>', 'Key file', 'key.json')
    .option('-i,--in <in>', 'File to enc', "plain.bmp")
    .option('-o,--out <out>', 'Output file', "enc.bmp")
    .parse(process.argv);



// 生成 RSA 密钥对
const { publicKey, privateKey } = crypto.generateKeyPairSync('rsa', {
    modulusLength: 2048, // RSA 密钥长度
    publicKeyEncoding: {
        type: 'pkcs1', // 公钥编码格式
        format: 'pem'  // 输出格式为 PEM
    },
    privateKeyEncoding: {
        type: 'pkcs1', // 私钥编码格式
        format: 'pem'  // 输出格式为 PEM
    }
});

// 将公钥保存到文件
fs.writeFileSync('public.pem', publicKey);
console.log('Public key saved to public.pem');

// 将私钥保存到文件
fs.writeFileSync('private.pem', privateKey);
console.log('Private key saved to private.pem');



function generateAESKeyAndIV() {
    const aesKey = crypto.randomBytes(16); // 16 bytes for AES-128
    const iv = crypto.randomBytes(16); // 16 bytes for AES-128
    return { aesKey, iv };
}
```



```
// 读取 BMP 文件
fs.readFile(program.opts().in, (err, data) => {
    // fs.readFile('input_image.bmp', (err, data) => {
    if (err) {
        console.error('Error reading file:', err);
        return;
    }
    // 解析 BMP 文件
    const bmpData = parseBMP(data);
    // 提取像素数据并转换为十六进制字符串
    const pixelDataHex = extractPixelDataHex(bmpData, 32);
    // 使用 AES CBC 加密
    const encryptedData = encryptAES(pixelDataHex);


    writeEncryptedDataToFile(bmpData, encryptedData);
});





// 解析 BMP 文件
function parseBMP(data) {
    const headerSize = data.readUInt32LE(14); // 读取文件头的大小
    const imageDataOffset = data.readUInt32LE(10); // 图像数据偏移量

    // 提取图像宽度和高度
    const width = data.readUInt32LE(18);
    const height = data.readUInt32LE(22);
    console.log(width + ' ' + height);
    return {
        headerSize,
        imageDataOffset,
        width,
        height,
        headerData: data.slice(0, 54),
        imageData: data.slice(imageDataOffset)
    };
}

// 提取像素数据并转换为十六进制字符串
```



```
function extractPixelDataHex(bmpData, bitsPerPixel) {
    const bytesPerPixel = bitsPerPixel / 8;
    const imageData = bmpData.imageData;
    const pixelDataHex = [];

    for (let i = 0; i < bmpData.height; i++) {
        for (let j = 0; j < bmpData.width; j++) {
            const offset = i * bmpData.width * bytesPerPixel + j *
bytesPerPixel;
            const pixel = imageData.slice(offset, offset +
bytesPerPixel);
            const pixelHex = pixel.toString('hex');
            pixelDataHex.push(pixelHex);
        }
    }
    console.log(pixelDataHex.length);
    return pixelDataHex;
}

function readkey(kfile) {

    const data = fs.readFileSync(kfile, 'utf8');

    const jsonData = JSON.parse(data)

    const key = Buffer.from(jsonData.key, 'hex');
    const iv = Buffer.from(jsonData.iv, 'hex');
    return { key, iv }
}



// 使用 AES CBC 加密
function encryptAES(data) {
    const { aesKey: key, iv: iv } = generateAESKeyAndIV()
    console.log(key);
    console.log(iv);
    const encryptedaes = crypto.publicEncrypt(publicKey,
Buffer.from(JSON.stringify({ key, iv }), 'utf8'));
    fs.writeFileSync("aeskey.pem", encryptedaes.toString('base64'));

    const cipher = crypto.createCipheriv('aes-128-cbc', key, iv);

    let encryptedData = cipher.update(data.join(''), 'hex', 'hex');
    encryptedData += cipher.final('hex');
    return {
        key: key.toString('hex'),
        iv: iv.toString('hex'),
        encryptedData
    };
}

// 将加密后的数据写入新的 BMP 文件
function writeEncryptedDataToFile(bmpData, encryptedData) {
    const { key, iv, encryptedData: data } = encryptedData;

    const bmpHeader = bmpData.headerData.slice(0, 14);
    const bmpInfoHeader = bmpData.headerData.slice(14, 54);


    // 创建新的 BMP 文件
    const encryptedImageData = Buffer.from(data, 'hex');
    const encryptedFileData = Buffer.concat([bmpHeader, bmpInfoHeader,
encryptedImageData]);

    fs.writeFile(program.opts().out, encryptedFileData, (err) => {
        if (err) {
            console.error('Error writing to file:', err);
            return;
        }
        console.log('Encrypted image file saved as :' +
program.opts().out);
        console.log('Encryption Key:', key);
        console.log('Initialization Vector:', iv);
    });
}
```

Dec.js

```
const fs = require('fs');
const crypto = require('crypto');
const { program } = require('commander');
program
    .option('-k,--key <key>', 'Key file', 'key.json')
    .option('-i,--in <in>', 'File to dec', "enc.bmp")
    .option('-o,--out <out>', 'Output file', "dec.bmp")
    .parse(process.argv);
```

```
let enfile=program.opts().in
fs.readFile(enfile, (err, data) => {
    if (err) {
        console.error('Error reading file:', err);
        return;
    }

    // 解密 BMP 文件
    const decryptedData = decryptBMP(data);

    // 将解密后的数据写入新的 BMP 文件
    writeDecryptedDataToFile(decryptedData);
});


// 解密 BMP 文件
function decryptBMP(data) {
    // 从文件中提取加密数据
    const encryptedImageData = data.slice(54); // 从偏移量 54 处开始，跳过文件头和信息头
    const encryptedDataHex = encryptedImageData.toString('hex');

    const privateKey = fs.readFileSync("private.pem", 'utf8');
    const encryptedaes = fs.readFileSync("aeskey.pem", 'utf8');

    const decryptedaes = crypto.privateDecrypt(privateKey, Buffer.from(encryptedaes,"base64"));

    const aes=JSON.parse(decryptedaes)

    const key=Buffer.from(aes.key,"hex")
    const iv=Buffer.from(aes.iv,"hex")
    console.log(key);
    console.log(iv);


    // 使用 AES CBC 解密
    const decipher = crypto.createDecipheriv('aes-128-cbc', key, iv);

    let decryptedData = decipher.update(encryptedDataHex, 'hex', 'hex');
    decryptedData += decipher.final('hex');

    return {
        headData:data.slice(0,54),
        pixelData:Buffer.from(decryptedData, 'hex')
    };
}

// 将解密后的数据写入新的 BMP 文件
function writeDecryptedDataToFile(decryptedData) {
    // 读取原始 BMP 文件头和信息头
    const bmpHeader = decryptedData.headData.slice(0, 14);
    const bmpInfoHeader = decryptedData.headData.slice(14, 54);

    // 创建新的 BMP 文件
    const decryptedImageData = decryptedData.pixelData;
    const decryptedFileData = Buffer.concat([bmpHeader, bmpInfoHeader,
decryptedImageData]);

    fs.writeFile(program.opts().out, decryptedFileData, (err) => {
        if (err) {
            console.error('Error writing to file:', err);
            return;
        }
        console.log('Decrypted image file saved
as :'+program.opts().out);
    });
}
```

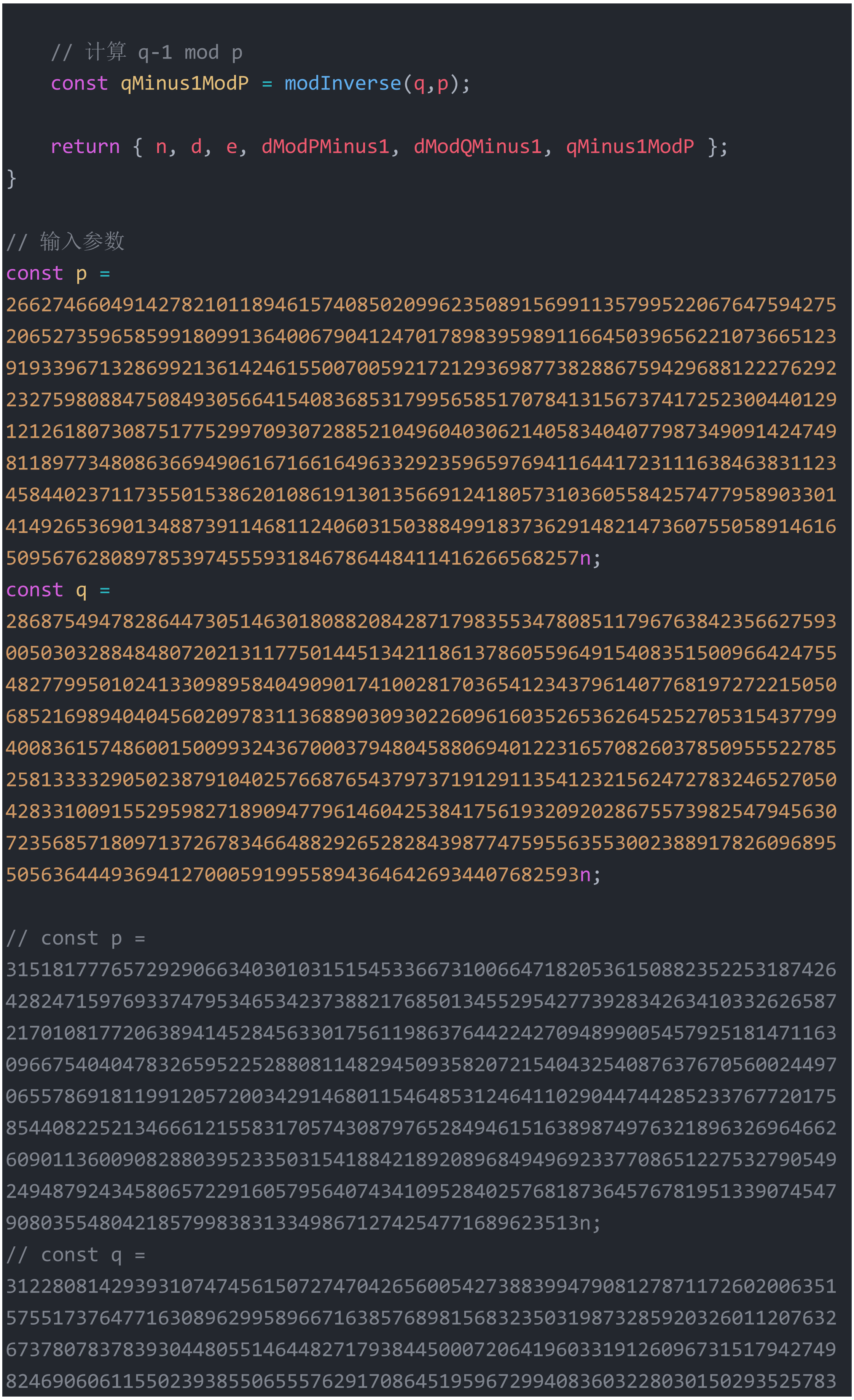
Gcd.js

```
// 定义函数来计算最大公约数
function gcd(bigInt1, bigInt2) {
    // 使用辗转相除法来计算最大公约数
    while (bigInt2 !== 0n) {
        const temp = bigInt2;
        // console.log(bigInt2);
        bigInt2 = bigInt1 % bigInt2;
        bigInt1 = temp;
    }
    return bigInt1;
}

// 两个超大整数
const bigInt1 =
76387674976617610690709707549913693254250630946279013423814566286402944
83576723293842407864298255066517354712931928578002920744325175601479342
96565646804636440319232118174807874707278193157951022791126432548954297
```

```
872603841823471708730019040711843329411852093347261870244367947957553762079472966281313126343653099184604229852078759250659887230164618217053184654632257026197939177514038547022906343467564156731888905345558825896885445137274880047279398158012899019023727313182644527939451811489338720718307850784366603399048017626264140284123069450132380165349204708385658847188179122993601433823666052262671276550074507090466095270790435243944044613783720793684122472997119531680070699442004485228493923284784588609625299335494520546908568332978745943998585734768939661601368054137698386418517184469311920103617525768647224332348763739953622815888621102511666165138133372157328365286393142987641723981258561805886361917214660712608451187202188877282119338253380439361187091882619543835302625472246720924991620467674200185807782910046428532568783970685234593602888422305491686373915683830020610512240092345516269641178633217279587340585381147938669982611180322446105897291197929327110789282688669532090634700770073422988590964225250401n;
const bigInt2 =
8257453230352727987205996083611924927467876120458769532622289211169342268346322375299634388183381967644903733559912417214496145427027254956721988308512463763281120693615792222840534071393074980762077238506791126339909131208264147978200838704703070369328468495474164128065966583172696020650361090028374095147747900931623331337880180057566279781633562179594032007854587452253456011800679608763699327977686789327009650791462970241997207731238948474262065077685273924666529377565303189508779580475499013651124403952110758192421559847639791002451557155022419866908364191649354978851250736301653152786792834185002549289578088732252348814786712647056246737399818451688819799945683784702174807918167354060334981848371709009902941634174705067621057569867546563489955228609184225558415642776670302913547498803162317655424566214364429366367792388796456095209075308157031922113847324082792128272456086838179448027864861814855945170296255542403540294916048693047962499625508963680441400589569308234136277898543659042895812833949323592040527735756576476005229995515153526350952978941824427177078891317284336320446076086808680961181640941049041256254665150119759613810158261696578922846119768440364563591963843063934695815733130597665757383432415903n;

// 计算最大公约数
const result = gcd(bigInt1, bigInt2);
console.log("最大公约数:", result.toString());
console.log("n/gcd:", bigInt1/result);
```

gen_key.js

```
function gcd(a, b) {
    if (b === 0n) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}

function modInverse(a, m) {
    let m0 = m, t, q;
    let x0 = 0n, x1 = 1n;

    if (m === 1n) {
        return 0n;
    }

    while (a > 1n) {
        // q 是商，t 是余数
        q = a / m;
        t = m;

        // 使用辗转相除法计算下一次的余数
        m = a % m;
        a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }

    if (x1 < 0n) {
        x1 += m0;
    }

    return x1;
}

function calculateRSAParameters(p, q, e) {
    const n = p * q; // 计算 n
    const phi = (p - 1n) * (q - 1n); // 计算 ϕ(n)

    // 计算 e 的模反演
    const d = modInverse(e, phi);

    // 计算 d mod (p-1) 和 d mod (q-1)
    const dModPMinus1 = d % (p - 1n);
    const dModQMinus1 = d % (q - 1n);
```



```
    // 计算 q-1 mod p
    const qMinus1ModP = modInverse(q,p);

    return { n, d, e, dModPMinus1, dModQMinus1, qMinus1ModP };
}

// 输入参数
const p =
26627466049142782101189461574085020996235089156991135799522067647594275
```

```
    // 计算 q-1 mod p
    const qMinus1ModP = modInverse(q,p);

    return { n, d, e, dModPMinus1, dModQMinus1, qMinus1ModP };
}

// 输入参数
const p =
26627466049142782101189461574085020996235089156991135799522067647594275
20652735965859918099136400679041247017898395989116645039656221073665123
91933967132869921361424615500700592172129369877382886759429688122276292
23275980884750849305664154083685317995658517078413156737417252300440129
12126180730875177529970930728852104960403062140583404077987349091424749
81189773480863669490616716616496332923596597694116441723111638463831123
45844023711735501538620108619130135669124180573103605584257477958903301
41492653690134887391146811240603150388499183736291482147360755058914616
5095676280897853974555931846786448411416266568257n;
const q =
28687549478286447305146301808820842871798355347808511796763842356627593
00503032884848072021311775014451342118613786055964915408351500966424755
48277995010241330989584049090174100281703654123437961407768197272215050
68521698940404560209783113688903093022609616035265362645252705315437799
40083615748600150099324367000379480458806940122316570826037850955522785
25813333290502387910402576687654379737191291135412321562472783246527050
42833100915529598271890947796146042538417561932092028675573982547945630
72356857180971372678346648829265282843987747595563553002388917826096895
5056364449369412700059199558943646426934407682593n;

// const p =
31518177765729290663403010315154533667310066471820536150882352253187426
42824715976933747953465342373882176850134552954277392834263410332626587
21701081772063894145284563301756119863764422427094899005457925181471163
09667540404783265952252880811482945093582072154043254087637670560024497
06557869181199120572003429146801154648531246411029044744285233767720175
85440822521346661215583170574308797652849461516389874976321896326964662
60901136009082880395233503154188421892089684949692337708651227532790549
24948792434580657229160579564074341095284025768187364576781951339074547
9080355480421857998383133498671274254771689623513n;
// const q =
31228081429393107474561507274704265600542738839947908127871172602006351
57551737647716308962995896671638576898156832350319873285920326011207632
67378078378393044805514644827179384450007206419603319126096731517942749
82469060611550239385506555762917086451959672994083603228030150293525783
```

```
304059251116128699302033650788176023836016857608350962781822536781996416
627056207053778817992702951569083785366683283330648466283312117252779050
258956055313372661579725165622531597301331397287817356686035735160405102
566121046574181753260238219660669540452666902834213164540378301783926
6710361200049839548767544661001788947463593677737n;
const e = 65537n;

// 计算 RSA 参数
const { n, d, dModPMinus1, dModQMinus1, qMinus1ModP } =
calculateRSAParameters(p, q, e);

console.log("RSA Parameters:");
console.log("n:", n.toString(16));
console.log("e:", e.toString(16));
console.log("d:", d.toString(16));
console.log("p:", p.toString(16));
console.log("q:", q.toString(16));
console.log("d % (p-1):", dModPMinus1.toString(16));
console.log("d % (q-1):", dModQMinus1.toString(16));
console.log("q-1 mod p:", qMinus1ModP.toString(16));
```

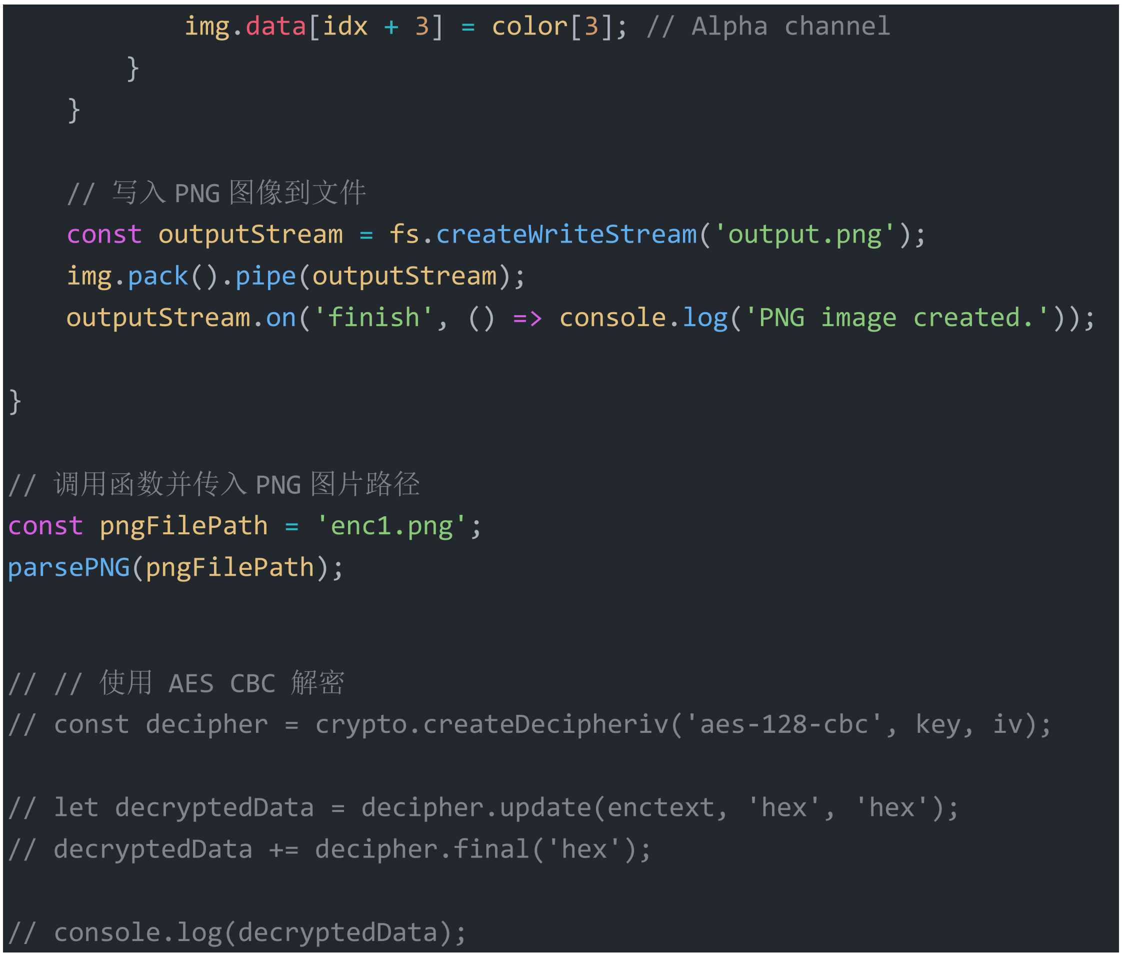
deckey.js

```
const fs = require('fs');
const crypto = require('crypto');

// 读取本地 PEM 格式的私钥
const privateKey = fs.readFileSync('private.pem', 'utf8');
console.log(privateKey);
// 待解密的文本
const encryptedText =
'MzhKNQx+U8ltsj5is29pSwu7yqdgoWPWIhgEwUTz3ywE84ue99Z7T/AISGOuyud6ET4E8x
XFS/7wadzwYj3yL6dQrw+F9KFPJRNkTDQll0Re+3kkGt2+M68HJRvmIcJaD1/0PNTv9gek5
PdL59TNq/VerwqXusAIIOdclwhb+U1EGJzJ0RS+8Wyp/+PU4J5P2mtFSak5SKNzDB8yg00u
yhRBZGriQzw+QQRZanWJYs45UFYIP+9ZMUK3lOkf3b8CT+qGW/HcDFwG59hn59PUvN8UFER
3PcOTIRD/+RBSKoi1Sdr7uxvQ3XTBvFJKlDMp1es4yzewmOgluBY2DtGV+aAbLzu5Sy6EfF
7tJgid8V9T9ZQ8nqW9vtWkt6Y2okRhdkpX+E+y240gU1BEHOUNglM6oJ1b0nGiAL5cjUtX0
IknEAsZR/U2ztsMQRzvy10xJpIgipKB52aNh6BnYzFH4DYndfehKh1NjVckcJOK+krTiUNw
QMNhRYSZ8v1pZH6jR96TuDPib1KcJopjaGdf9zNa2bkdJ7NSWTe9j1jHMPJYjrP6XCefsix
RTWp5dEz3KgzWEgGBHmIhz2SYYWLcy0SKb3ljYFUrY6tDwVRC+Srkk4GOeS09OvxT3r9E/J
daiA9BXuRjrV7LeCAW18AwbpZEaTHxjrVcoZ5sWpNasCI='; // 替换为实际的加密文本

// 使用私钥解密文本
```

```
const decryptedText = crypto.privateDecrypt(privateKey,
    Buffer.from(encryptedText, 'base64')
).toString();

console.log('Decrypted Text:');
console.log(decryptedText);
```

decpng.js

```
const fs = require('fs');
const PNG = require('pngjs').PNG;
const crypto = require('crypto');
const { program } = require('commander');
// 将数字转换为两位的十六进制字符串
function toHex(num) {
    const hex = num.toString(16);
    return hex.length === 1 ? '0' + hex : hex;
}
// 读取 PNG 图片并解析像素值
function parsePNG(filePath) {
    let enctext = ""
    fs.createReadStream(filePath)
        .pipe(new PNG())
        .on('parsed', function () {
            let hexString = '';

            for (let y = 0; y < this.height; y++) {
                for (let x = 0; x < this.width; x++) {
                    const idx = (this.width * y + x) << 2;
                    const rgba = {
                        r: this.data[idx],
                        g: this.data[idx + 1],
                        b: this.data[idx + 2],
                        a: this.data[idx + 3]
                    };
                    const hex = toHex(rgba.r) + toHex(rgba.g) +
toHex(rgba.b) + toHex(rgba.a);
                    hexString += hex;
                }
            }
            enctext = hexString
            // console.log(enctext);
            const ivhex = enctext.slice(0, 32)

            enctext = enctext.slice(32, -1912 * 8)

            const keyhex = "3c63655e71473b74515d557233505061"
            const key = Buffer.from(keyhex, 'hex');
            const iv = Buffer.from(ivhex, 'hex');
            console.log(iv);
            // 使用 AES CBC 解密
            const decipher = crypto.createDecipheriv('aes-128-cbc', key,
iv);
            let decrypted = decipher.update(enctext, 'hex', 'hex');
            decrypted += decipher.final('hex');
            // createnewpng(decrypted)
            // console.log(decrypted);
        });
    // return enctext
}
// 从 hex 字符串解析颜色值
function parseColor(hex) {
    const r = parseInt(hex.slice(0, 2), 16);
    const g = parseInt(hex.slice(2, 4), 16);
    const b = parseInt(hex.slice(4, 6), 16);
    const a = parseInt(hex.slice(6, 8), 16);
    return [r, g, b, a];
}
function createnewpng(hex) {


    // 创建 1920x1080 的 PNG 图像
    const width = 1920;
    const height = 1080;
    const img = new PNG({ width, height });

    // 从 hex 字符串读取像素数据并写入图像
    const hexString = hex; // 替换为实际的 hex 字符串
    let index = 0;
    for (let y = 0; y < height; y++) {
        for (let x = 0; x < width; x++) {
            const color = parseColor(hexString.slice(index, index + 8));
            index += 8;
            const idx = (width * y + x) << 2;
            img.data[idx] = color[0]; // Red channel
            img.data[idx + 1] = color[1]; // Green channel
            img.data[idx + 2] = color[2]; // Blue channel
```



```
            img.data[idx + 3] = color[3]; // Alpha channel
        }
    }

    // 写入 PNG 图像到文件
    const outputStream = fs.createWriteStream('output.png');
    img.pack().pipe(outputStream);
    outputStream.on('finish', () => console.log('PNG image created.'));

}

// 调用函数并传入 PNG 图片路径
const pngFilePath = 'enc1.png';
parsePNG(pngFilePath);


// // 使用 AES CBC 解密
// const decipher = crypto.createDecipheriv('aes-128-cbc', key, iv);

// let decryptedData = decipher.update(enctext, 'hex', 'hex');
// decryptedData += decipher.final('hex');

// console.log(decryptedData);
```