Testbed Charlie Mk. II

April 6, 2020

1 Pictures as Secure Group Messages

1.1 Test Code by Charlie Cook

1.2 Done on April 5th 2020

Imports of what we need from PyCryptoDome *(version 3.9.7)*

```
[1]: from Crypto.PublicKey import RSA as rsa
     from Crypto.Random import get_random_bytes as grb
     from Crypto.Cipher import AES as aes
     from Crypto.Cipher import PKCS1_OAEP as pkcs1_oaep
```

1.2.1 RSA Stuff

```
[2]: myKeyRSA = rsa.generate(2048)
```

```
[3]: myKeyRSA
```

```
[3]: RsaKey(n=179302984628135057114397032527929829893606024986746236253651017476739822
     905410912629942556186799032953152739594418718067009559480816726509127219408073434
     692860091479486596443288519913274239629696269125549930772498148135146362438126835
     219024485960645874602463014006644739777908638669824947276787277316561444251463228
     101314988272949146218440917471920011718534138207813521887142428454044636925498188
     552093774279572830744190522579652745076748130637142846091045578831866438040970536
     333220671450240380428103966500657275423844260994253649944238673131632600786755719
     1944987308043164152887560355190641464195171479099201311637832647, e=65537, d=1
     630963973068428196506745059904835984668838544569210109892271946408906471562469859
     132462959064924675293770922820108508223847911637326176507068230638089150816072143
     427289968098015377013315711607594541573714692227641446763281090247255802423766061
     928926733013338139515821911699655358609099038103853168407915579122753524407841755
     387376810971919262183363978209738442453797304888418870720778761884641205146622124
     911086424740569343813595162055775820849090102956142531002718636379510366833620886
     882336016333151161923094258290746690245902968392082742815485481827135450561332324
     57640244811720845099854677209545712246357244 6817, p=1337638393370608093607486769
     587077933745136971558929265667624682531325169693, q=1340444364611304215599376839
     6517779768702127159822222873711898144141612260180264907741656515529038096030101O
     88287074521878642400947909004051167338011130729181641802767470377075274292677946
     4808161074065448670098590280280084396925036399161567990911033323135301081840521O
     03999202028254892420659387720262898083379, u=77573041635623588726618013254852803
     10553663365096960562207344670792078827699546686471525303547031789763677167152633
     52822313624673578395755676716394705746573378721124852259402186618961267861564046
     77278352142647852767995124155067717100631593552676981192376929183781941476728943
     114157287970800326262109877715993)
```

Public Key Encryption Object (U for **pUblic**)

[4]:
```
myCipherU = pkcs1_oaep.new(myKeyRSA.publickey())
```

[5]:
```
myCipherU
```

[5]:
```
<Crypto.Cipher.PKCS1_OAEP.PKCS1OAEP_Cipher at 0x7fa0c80a0a60>
```

Ciphertext (the argument for any Encryption Object must be a `bytes` object)

[6]:
```
c = myCipherU.encrypt(bytes("Hello World!", "ascii"))
```

[7]:
```
c
```

[7]:
```
b'7s\x12S\x0f\x80\x17*~eeep39\x838?\x1e\xe2\xd7\xd5\x99\x84l\xde\x14\xf9!\x9b\x9
d;\t(y\xab\x93\n\x0c\x14\x87\x9eWO\xbd\x947\xe1K\x1a\x1d\xd0\xa14O\x93pK\x12m\x0
eYr\xc9\xa2`.\xf5\x89\xc1\xfa\x9c]\x823\xa1\x91\xdd3\xf1\xc0H\x81\xec\xf6\xc9\x0
8\x06\xe3\xd8\x82\x0e\xbc\x89\xa3\xa5\x88\xb4\xbe4\xa7\xcc\xb2MI1\xc1=\xa4\x0ft\
xce\xcf\xc1P\xbf\'x3\xdf\xde\xd9\xed(\xb4\xe3\xab\xcc\xa4\xc7x\xb3o\xc1iY|\xef<\
xc75\\\x89\x8aS\xff\x1a"\x00l\xb0\xe0\xe1\xdd\xa2\xbf+\xde\xf2=p~\xe8\xb0\x05r\x
0c\x81A?}!"\x02\x14\xa9\xa9=\xe5p\x9d\xbf^\x17\x94\x8f\xbb4\x9c\x02\x1d\x0e\xa5\
x1dr\xbbof\xe9x\x15\x14u\x82\x15\xf9\x82 \x93\x81\x17\x12\x8a\x83\xe4)6A\xa3R\x1
0!\xc2[MbYC!\x92\xaf\xb2\xea]p\x83N\xab\x89\x8e+\xc0E\xf4\x83N\x9cm\x86($\x9e(\x
a3g\xb4'
```

Private Key Decryption Object (R for **pRivate**)

[8]:
```
myCipherR = pkcs1_oaep.new(rsa.import_key(myKeyRSA.export_key()))
```

[9]:
```
myCipherR.decrypt(c)
```

[9]:
```
b'Hello World!'
```

Raw key data & binary text key data (the latter can be stored in `.pem` files)

[10]:
```
myKeyRSA.publickey()
```

```
[10]: RsaKey(n=179302984628135057114397032527929829893606024986746236253651017476739829
      054109126299425561867990329531527395944187180670095594808167265091272194080734
      369286009147948659644328851991327423962969626912554993077249814813514636243812683
      521902448596064587460246301400664473977790863866982494727678727731656144425146328
      101314988272949146218440917471920011718534138207813521887142428454044636925498185
      520937742795728307441905225796527450767481306371428460910455788318664380409705
      36333220671450240380428103966500657275423844260994253649944238673131632600786755
      7191944987308043164152887560355190641464195171479099201311637832647, e=65537)
```

```
[11]: myKeyRSA.publickey().export_key()
```

```
[11]: b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjgkK8y
      5NivgGOmJITntA\nazU45ejFHJu43dgQnKb6eFbNv/B0I73kBexHByc3sMQeKh+EMmkggAOi77KtyN+K
      \n+upeYNLHmVNXQYPL2CxEhoUOKw21SlNR5cQeDboATDs4cM4QXkSJeLjYwGkgMktv\n4Gvz+LSq3Aeg
      ay6cemuu9lRmXOKVCPDoa4duC0yT5H8pFnaf0WYkCbPrEgY0bO2o\nW3fPCn2r6s8UAPixg9KpZK2jbx
      BtcCOByXDfRvgxGP8ACXtsEEUIp1KZz74Mwuk9\nxCYWH6fFAprbbh7THc17nn3pJNp8HrUjIQ9OntbK
      vEnYyB4Idgl/+yLj7/PviWkD\nxwIDAQAB\n-----END PUBLIC KEY-----'
```

```
[12]: myKeyRSA.export_key()
```

```
[12]: b'-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEAjgkK8y5NivgGOmJITntAazU45ejFH
      Ju43dgQnKb6eFbNv/B0\nI73kBexHByc3sMQeKh+EMmkggAOi77KtyN+K+upeYNLHmVNXQYPL2CxEhoU
      OKw21\nSlNR5cQeDboATDs4cM4QXkSJeLjYwGkgMktv4Gvz+LSq3Aegay6cemuu9lRmXOKV\nCPDoa4d
      uC0yT5H8pFnaf0WYkCbPrEgY0bO2oW3fPCn2r6s8UAPixg9KpZK2jbxBt\ncCOByXDfRvgxGP8ACXtsE
      EUIp1KZz74Mwuk9xCYWH6fFAprbbh7THc17nn3pJNp8\nHrUjIQ9OntbKvEnYyB4Idgl/+yLj7/PviWk
      DxwIDAQABAoIBAAzrcvzegAcyuxOh\nz69JNCQ3BHnv9nNnvCUnJjuu+fxASRmQfGqpNziitBieefOw+
      IgfCHNdb1YK/qM3\nFD/ACxr6fqM9Xmfqq3aLeoYAtisA/LebLkZt1cvvTvHeCBuRt3dAsenL6i/6Vce
      1\ncvb1jbOoSNpH9NLBjtdbt2LGBiA54nT9A7742tasOzyuu3GvKWJ1wlU5FQt5bUbP\nRlrZvlmAvfA
      zbfDLJiLslZuaGguYhswjaC/OLxZXFxCLarM2Qmr2DAYMOQf5VvcX\nL8qT5mRybI2YhNyDPFE+epuAH
      Y+OftDhALW9UH8YUBxA948LQioa60b/Gb4J49Ak\nSjayTmECgYEAvnxsePMQUIza/jHSq6bpAGxXXOG
      BOqYkdYDRlic85r3oRX4q+3z4\ngYjZdNkXSyN7XxRDbiz4L0Yi/vDzMxnB9zn9apmdm+YGOlC1Xq+rk
      Cb/ExwH6fBB\nx6bIpNoCpVVW9db1HL+Z/kQB6r78GTMPD9dqS+YuemLzdVXHVB8zUBOCgYEAvuK3\nl
      MkgPla53iVuGOv1/cQYwd1AO+SvbyToPlTNZBakky8ptw+3eEVGUj7Bid6aM6oC\nHlR5TD/Rc60d3+G
      JofruVP+pWpiLHbA9k7fQN3nYsoI7my9Q99+bwWPMDRbZQfmQ\nGsX4AZM2Tm+Geabh+7oIE0haVOesz
      fbBfa8u5jMCgYBfjvGnt1/u2MqfjIUAAYqP\n4Mnu2V3H+OL9A3Og3OxLYOedvQ54/Rd5fBqC6Mkrs5A
      XgYXWRo72wDucI2oNr5VD\nnf7INGoBpGJpbMWfy4bImReLEZvSuul06/Fp6cg8AtTOHVj3ZBMvoBRKH
      CdByQB8\nVPeJPd8BHJS/vxlw1Pob7QKBgFEPyHQ3wSiX/dCXxp1y8aD1gKn7Qod5awLhAntN\n2fcyI
      muasLVbyEcb7BQLMHdS/9sGzGWC7Av0YgcksjYb+i2+eS3BcHLXquRNrqin\ne5XDecG8yoFJY7IP1H4
      Y2lOIq3R3/blQ7tQEZyCB/fJ1ayxSQRGnPS0nQVwp+Ntd\n5KW/AoGAUD/lfzdknRkuqquaDMSFMB67C
      OgJ4BcM2uIg+ncAeOxdD5173WPK2Nlb\nPwznEktLUTZjaHci8dAazXgaUCmmd8joJLFETYz8cfbrZcl
      YYd7gpG1vFeP/3UH6\n9tW5ZfdYC+kGKaZNNfZ1YDoRN4tAjbEvPUGw8lMpPHG7QhGxzkE=\n-----
      END RSA PRIVATE KEY-----'
```

1.2.2 AES Stuff

[13]:
```
myKeyAES = grb(16)
```

We want AES keys to be randomized, as they will be generated per message/image. These session keys will be sent securely by being encrypted by RSA.

[14]:
```
myKeyAES
```

[14]:
```
b'\x8a\xbe\xe9\xbe\x010\xc6x\x8f\xfa\x11\xbb\x94\x19\xf1\xd6'
```

AES Encrypt/Decrypt Object (source of the nonce; an explicit nonce can be generated elsewhere) (S for **Symmetric**)

[15]:
```
myCipherS = aes.new(myKeyAES, aes.MODE_EAX)
```

[16]:
```
myCipherS
```

[16]:
```
<Crypto.Cipher._mode_eax.EaxMode at 0x7fa0c804f0a0>
```

[17]:
```
myCipherS.nonce
```

[17]:
```
b'\x90\x12\x06\xe5\xb9\x86\x9d\xf0 \xe5\xb8{\x8bMZ|'
```

Ciphertext and Digest/Tag/Fingerprint (Like RSA, AES works on `bytes` objects only)

[18]:
```
c, t = myCipherS.encrypt_and_digest(bytes("Hello World!", "ascii"))
```

[19]:
```
c
```

[19]:
```
b'\xe7\xbc\x97e\xb2\xc8+\xe4\x1f\x19\x81\x0e'
```

[20]:
```
t
```

[20]:
```
b'\xdb\x82P"B\x97\xb4?\xd3\x01\xefm\xd5*\xf6\x8e'
```

An identical E/D Object as seen above, with the original's nonce provided

[21]:
```
myCipherS2 = aes.new(myKeyAES, aes.MODE_EAX, myCipherS.nonce)
```

[22]:
```
myCipherS2.decrypt_and_verify(c, t)
```

[22]:
```
b'Hello World!'
```

1.2.3 Imaging Stuff

[23]:
```
from PIL import Image as img
```

```
[24]: len(myKeyRSA.export_key())
```

```
[24]: 1674
```

```
[25]: 1674 // 2
```

```
[25]: 837
```

```
[26]: 837 // 3
```

```
[26]: 279
```

```
[27]: 279 // 3
```

```
[27]: 93
```

```
[28]: 93 // 3
```

```
[28]: 31
```

```
[29]: 3 ** 3 * 2
```

```
[29]: 54
```

```
[30]: int(len(myKeyRSA.export_key()) ** 0.5) + 1
```

```
[30]: 41
```

```
[31]: img.frombytes("L", (31, 54), myKeyRSA.export_key())
```

```
[31]:
```

```
[32]: img.frombytes("L", (279, 6), myKeyRSA.export_key())
```

```
[32]:
```

```
[33]: 100 // 6 == 100 / 6
```

```
[33]: False
```

[34]:

```
def bestBox(n):
    ub = int(n ** 0.5) + 1 #Cieling of sqrt is the upper bound of factors
    best = [1, n]
    for k in range(2, ub):
        if n // k == n / k:
            j = n // k
            if j - k < best[1] - best[0]:
                best = [k, j]
    return best
```

[35]:

```
img.frombytes(
    "L",
    bestBox(len(myKeyRSA.publickey().export_key())),
    myKeyRSA.publickey().export_key()
)
```

[35]:

[36]:

```
def bytesToGray(b):
    l = len(b)
    return img.frombytes("L", bestBox(l), b)
```

[37]:

```
bytesToGray(myKeyAES)
```

[37]:

[38]:

```
bytesToGray(
    myCipherU.encrypt(bytes("Hello World!", "ascii"))
)
```

[38]:

[ ]:

```
```