Testbed Charlie

April 5, 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=2342773793381822845444130880265347872131473271180555737391387850044962 2
724853215130996536697510177077210715847694368409436887453450539104539321130660 32
```

```
RsaKey(n=23427737933818228454441308802653478721314732711805557373913878500449622
72485321513099653669751017707721071584769436840943688745345053910453932113066032
78220912164065519387110674168018374801991048298811619330441493763240468727149465
28820196924461007100812806857081926849949300063720865487269958040670568347292173
42201950078565481391793105112253776770078358087854649204889880607639464478969044
71498866166872565548255546687404115306703138796813629213405602664658778831134841
53412580765257930120209208728113181368809673407346166829260528692434840458467839
49013389416345681848426423757844473638227713022570658231196376139 9, e=65537, d=6
18309922326740555971415390679915676563885784831769520592637825649801345394018306
78212830880968159896265419805446326846701359026684594154678292337151248439218213
59618638325003627761658029548354617099417086051446313469006272411912738252081615
01788637696352975069544015463971588724168897027347464145137246410920484876825981
91046458719213655471648514142328995756176473782125775885786448846067862046160438
67143963406721016534672790345682646382098535296141195048390276543616933126758213
44115204044521338861088389782409233849447215624051100996158100973761118017336679
96740157711998551914838892526606505727281178655897273 7, p=135391988390647555909 8
60448706456299124830993006460548452562557172811757345206833423113375240814455435
69888126760857978614424573408065920798003506109400051328752143968387277398828680
58326660139993487871964898032904039294695152159512868795616180152299224660671993
```

1

```
72865671775241359367815835613768685103734546967, q=17303636804728795468494928555
14191076233784869587220689342892371219470802375399311261279400711751128980945125
65096014593193344869674113327484448572115776759659898561888564242900432602794853
57269593907544450719696891254113610674725718233385859241834074525971939527652913
5680774412862315155283461351020688996897, u=876557880457741085866637443349167407
19445206744479813833540333336894956119986682952948374494801195404580781184524061
42368197296036180628328042444415155970462493785996320972674702517525367757303256
77359474648245724479815324650608005104632174805518437980048565585409261038680499
34025496508217391534215255996441)
```

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

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

[5]:
```
myCipherU
```

[5]:
```
<Crypto.Cipher.PKCS1_OAEP.PKCS1OAEP_Cipher at 0x7f40642804c0>
```

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

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

[7]:
```
c
```

[7]:
```
b'e\xfb\x1eF\xa9\xb8\x8e\x05,\xd2>xR\x1d\n+EtN\xa5\x86\x81\x89\xaa\xe8\t\x90m\\\
xada\xac$F\xc0\xa3\xc4\x80\xfd\x12\xfb\x17\x02#\xf2\xdd>\xb6\x02\xcd\xbfq\t\xef\
xaa\x189\xe6\xc9W\xc7WG\x00\xa3)\xc1\x99\xdb\xe3\x8f\xc8\x9fR\xa0^\xd3\x9c\xb3\x
d9x\x00]\xbf\xa9\xc7B\xa4\xe4V9p\xc1\xeat\xa1U\xdd\x89"\x80\x80\xb2\xd1i\xceT\xc
c\\\xc1p\xd2\xa2\x7f=\xabn\xa6H\xc4\xd3\xcf0:z\xeal\xca.M\x9d\xa4\x18\x00t\xcf\x
c7e(\x9d\xd9\xc6c0o\\\x8f\xd7\x8a5B\x1c$_\xc9\xff\xb4\xf8D\xb6+mP\xb5\x86\xea\xd
e\x0f-\x93\xf0*\xc9`\xaee)\xd3\x96\xfdu\x89\xc3\x02\xc9\xe3$\x18R\x90U\xa1\xb9D\
xf1\xbdnl\xd8\x08]7\xc5\xd4\xd7\x05\xdd\xc9mh\x87\x0eG\xe1\x88A\xefM\xb3p\xaa\xa
0i\xf1C\x93\x1b\xbd\x88@._&\x02\x05\x7fFe\xb6Tt\xe4\x9ch\xe5\x1c\xa7\x95,\xaeX<O
Z\x84:'
```

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=23427737933818228454441308802653478721314732711805557373913878500449622
      72485321513099653669751017707721071584769436840943688745345053910453932113066032
      78220912164065519387110674168018374801991048298811619330441493763240468727149465
      28820196924461007100812806857081926849949300063720865487269958040670568347292173
      42201950078565481391793105112253776770078358087854649204889880607639464478969044
      71498866166872565548255546687404115306703138796813629213405602664658778831134841
      53412580765257930120209208728113181368809673407346166829260528692434840458467839
      490133894163456818484264237578444736382277130225706582311963761399, e=65537)
```

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

```
[11]: b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuZVb0x
      yWdbwrNUFOT7J9\njqL6jvvwSt7YSnNBcEFZ10mTX/ryX9SMezsRnzJ0wd05hHCUAo4fIEdguDiMeEWH
      \n4g1c0FvMkKutpRlJnf4+z3CeHS/M27ZWmQSyNU3wZ0ebAL0Lnjjf0qczsJo63qUm\nAQSKV/N6stUk
      GL3t28YagDGs44Pp36e+ByTnTEtzSwYEUOLKeeyePIqblVePcsnL\n0PfQFY+8RZLLkiRtv9RBj5Bak2
      lCdZS8s22sDE/j1VfL2bwNQEA2VIzSQRWFieVZ\nqUT3rLW1FSXr4vTf9sl3r14iIYl0T2sa7Vk1Dc2H
      q45XckNSk80esgrBi/CxDlCa\n9wIDAQAB\n-----END PUBLIC KEY-----'
```

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

```
[12]: b'-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEAuZVb0xyWdbwrNUFOT7J9jqL6jvvwS
      t7YSnNBcEFZ10mTX/ry\nX9SMezsRnzJ0wd05hHCUAo4fIEdguDiMeEWH4g1c0FvMkKutpRlJnf4+z3C
      eHS/M\n27ZWmQSyNU3wZ0ebAL0Lnjjf0qczsJo63qUmAQSKV/N6stUkGL3t28YagDGs44Pp\n36e+ByT
      nTEtzSwYEUOLKeeyePIqblVePcsnL0PfQFY+8RZLLkiRtv9RBj5Bak2lC\ndZS8s22sDE/j1VfL2bwNQ
      EA2VIzSQRWFieVZqUT3rLW1FSXr4vTf9sl3r14iIYl0\nT2sa7Vk1Dc2Hq45XckNSk80esgrBi/CxDlC
      a9wIDAQABAoIBAATl4JQ0IfMeMem9\nkn/fTxmvoCXxQkic6bUf+5nR2mQoiJ+KM6a8floA7MMjMBpZv
      gXfzsLj1gXNEcwV\nj1IxS5fMuyazSt/amBeju4ypmXtyFnTgvmsMvJZcnbNzHiWA/0ydW1b0sMw80Mw
      Y\nFfZVrNNut5QAqhFZjL1Wbeup+9MRm6NHVJRl89n0QsRW2nNxCGxkdSMioAHZ0Rsz\nG0fP0hqfHxD
      idV5jmm02oi4XKPxQNWdpp/LW6wwfUnz8ty0ekW0wS7LGWSNeB36Y\ncHl3rFEJKRKBSTg2Kmnn/M2SS
      2vGikr0hn9PaCTBFqjc2XFDWoge3i6vw4Uwr75x\n2jpHI0ECgYEAwM35sz8Brr54zkF5A70s4nxTAgt
      WqgRMLfcG0J3VhWGARx9UNVfK\nYEobDYiVCmfzeZ8Y0v3EmWvxX14JQTAfDJ2HVdy8xdfs2WPARz0gK
      BCYeceG4fRK\nHnjD3f8yBJPFaYGYKjGbQKGZUaqx7TyZ4bMVyhxGJKslb6/ySlWPYhcCgYEA9ml2\ns
      pj45+agd/kRd7/aCkiqMpAR9h9nrcBsf9fM964wGDrtluL7jrstCUdg5mv2ahzc\ncjSKJeNUce6tblm
      VvKgLV0YBDz1zfga8JW/BNH0aEuM0r+D2QI1oG9WvFRCNHgio\nmeprPZVsAa+mYxyxsMez+c5qFcKV2
      cIabDpaeiECgYBcBrRlgvBZqN5ejE+nojb3\nT6IL0sA93FlEI0Fkd8F/rX8d820tHN8iwJxTBFsnIWD
      ML0zQc2nQVP5bp7XBIKfh\nTJtZN0f/IsYRKRSQp6qNFQbCMaNG3GQ4USokHvePybyTVGD92rmgysE0g
      sX0V1WX\nrfP7iS1NuUbgHVoHcvlCNwKBgDzpBPZQ6fYpBDGj17WKLx0rsqadBl0Yn8BLeIaB\niaEP5
      c0XjXNm2FgH3LIWetV6iobga32vjxaegxlGrohAjMr9nxALtedWgV39vDRk\n05VrngJdN4DiUvzR95I
      p2AWU+c0axtKFJ80pSnGAQdjuFt5j/xEY/RAz8WbKnE10\nfzkBAoGAXyJ9i6uHr3erz/XChrkv4p7t5
      DLIVFpN9ilYgm2MGJTnprvAQI+r6goQ\nItTSliAunLN3oY3gpErHMaHBrnKHi3r7Cir5j8MgAkV0plZ
      v5vakUqbsVpwyuZdT\nakFXdHrjfd0L/N00Q4oZzpBqi2a6C5retLLhAnwzztnxVLJIYZk=\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'\xe5\xd7\x1b\xe5\xe0j\xea|g\xc1\x972\xc6Q(@'
```

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 0x7f406421a100>
```

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

```
[17]: b'S+\x1e(\xda\x97\x9f\xfd]\x8d\xd9j*\x11\xde\x13'
```

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'\xa5\xbc9\x8c\x1d\xc9.]\x156\xc6a'
```

```
[20]: t
```

```
[20]: b'\xe6,\xa2@\x18C\n\xa9\xa2\xc8*\x84\njob'
```

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!'
```