Group number:22
Group Member: Hew Sook Mun (30305837)
Chua Jee Ann (30327334)
Chin Wen Yuan (29975239)

1 Task 1

(a) Compute the Alice's signature (sig). Given RSA-3072 digital signature scheme: sig = $(msg)^d$ mod N
By using the following command line in sagemath tool according to the signature scheme given:

Steps:

```
d=0x541af9701e04a45700ce962015c835a0d503fe1e5cca2b48a99e47a32473f2ea40f48c2e
ec31c98555657255d5565bcf3f4fb98886d6febc34a0950817dae88a3e808f569b3a47b1751d
4013a861095166ae2322e6dfe8740d844c8284ab3b29d7c4261efcf2c64c56bd6ce2bf4db342
6ee879683cf669f6c7351c55398cb03a8e4c9a0e3ccbe5d527a3912a8cea045414b7bdef2ffe9
a348c56dec274ba676e05a224553543910fe6940169f73be36bbca1c0cd53525f53e4b2aa9e
69423ef077b2d1bfe8d45927a677f74418240b95ed5c698e62fb429ece5fabbdcff8f64c480bff4
6bb6a448ae350739795abd156a5814378248b7100bfbaa07b039bc105a32a6fe74e0768857
7edecd515bd452a41cbfc017b9d26e76a5bec2ce433714a02f0f2c3784b65738adc849c3c31f
8a731132e4bd8c2b2c0b33de87403c2b7ff12ab3d9582453844b4ff03142f899b256e407c330
1adc46794d14bd668beac877e9cb5aa0602c447b75d3424d3a71495ed55a86fb1b01b5fbae2
a766f6172301

N=0xe55c85be1e8f31b8cfa79da46e313545fe58d51308f427be1798373cce2304c0cee692f4
ab78387dc5d3161b5a1f33df90858c5c0a8fe906579257043a527f33e37b3466b7929be81abe
c6e9979215abf92d71032caf5fffe4a5f1c176172d8fb62da7beecc255e45b75a44e30ebbeb91
ecb97de7dc51a0c1d19f1cb0e5658b4a66cd4500252dc8f50076c357f5dece3f94ef1133cd2c5
92a5c9eb22a2e818f95252f0917caf47737807ece3a0f508f1af03b8eabd2f3d6cc881b27627e
3cb5eda7862c25213592ebf1f8470dff22d7603d299ee69628101c75133d65618692aad5f3b2f
fb3a22e1084a900cb0543107b02f8062737181eab4870cf25f0ed473cf4095530702314dd0a8
cace3a6fd0169f2dfea254d3ab152381c3ae535f780a1b532fe040eae7ba864bf28543a6dec71
1e62878ec4471341c8ee00824e9cae7627c29de36f3678cbfe046dce37bd6c7639c51f9387e1
b756bda7622efb9ee49fb258266b19fb359ef3f959ffabb0ebf3747bb923cf69899bfdcaba18bd
4dbb7

msg=0x6b948843b86adb04a834cba6a76d5753da8ffbdcd01782a49d395f52f4c37a0cc39ee
b41646ebc2b2003bacb203328e210604f248e02fa95aa6eec50751abe267f5c0b70f60901a4f
d338f61bb2000acb3f2cf80d602acf85c5ee2f015667e9520e2d5c1aa84dcc69c9358a376846d
2a0e9b52877fe17a76ce4bf6c46c7a46f61102d42869e0a594c4ad71a699a603654e4d6bdf83
fc09b9741b70e82013302517efceebc9be49a7bc86ab89653f3281ffcc20824970410461510f4
a9b538f8d5468b872cbef23a348b61576ae1f840138f14e7f8f13643aae1467cd534803555f8b
2facb34fae15d53dc8c954bc8af0561597bfbb5a82c3b08bc83d349962aaaa6e164a138045b9
6dd9730aa7e1bb440838c42296ff2bdf53ca69f09c7e74c5e855455ffb052399e82e7e182d8ef
a08c96bdd166a00381d3fc53bb2a3d46b0aa6e2af8a45cd00e8bffe34fb7bafd20dade1efece7
331b417136e2ed971c8f16e193948f3c6595e9f63a948610f1d3e2246e6603d0b039f9bdd50fc
50baadef

sig=power_mod(msg,d,N)
```

```
sig=hex(sig)

print(sig)
```

Result:

```
sig=0x274e84c26582b994d2fda202d6774579e9124d4f07e0620c35df8f499fdf9ed1736703d
0d613bb805ec1030f6a9130fbdcf5a0f2dacf23bd9382b8112309b0fbccae7115d4a68d4660f2
b86758b8b16a0affa96ac9776934d4cf55d9ad64bf3a4b3adee1cc0633401319ac77d76cf6b7
9835b1c831a787b4e1f78d615b6a50856f15d08dcd94215a99644b822437ba1bb27dc4ef5e3
d30e661bed7c598159854683ded1da06939ece2239daf6d3207d8f093a10210c12e3c405f3f7
d4d89e2fca7c9d8b2272ddd6eaba0ff8f553f9e0a3440e82db55bc053727474f0701e45da59a5
ef02f5307e9ae8ade39862e8ccdcc14d3107485bcf5c7bb01600ff8932f126561a474c41fc23d9
af4651f2c8415d19a2514c37f37b360243149573afabdcc4aa1db33f9346e89006d5577dcbb1
ddd31db28b89f48c4f8f4bb0c61bb66b9c2998a1b7229f372e1e2e3d14905f1c6f9e37dd0acba
be1fad962fd2d12d162a8aeb9879702984ecdbf4978a5bfdf2b6204eeb077f0dab98d1e815ed
d26a66f49
```

(b) Do both messages pass the verification? Show how bob verifies the digital signatures.

Known: Alice’s public key (e,N) where N is given in (a) and e = 10001 (hexadecimal) , 2 message/signature pairs (msg1/sig1) and (msg2/sig2) received by Bob.

How it was done:
by using the verification scheme of digital signatures: msg’ = (sig)^{e} mod N
we can compute the msg’(valid message) and compare it with the msg given to see if the message given passes the verification.

The following command is input into sagemath for verification of **message/signature pair 1**:

Steps:

```
msg1=0x9cca26cb7a1713c2c95ee703089b84bd27311c5750c2a817586e7b1fed6e12a8051
a4626b48656a229eec292e30f0751d59ce1544300919801303cef1a08d08015ba5ea047e434
8b340aca99b3fcbcde9663b1a8d59eed23b6e1e834889729e53a691343f589babff5c8f8fe661
bd0a643644dde1ba9f37023bb01af97dd12bfbc5517a2a1a3e67108f0b287e7dbe9c9d81fdc5
c1c7875c8577c6acd7eeaee2a46a5ce3c2d5123b085cdb554c36fa3a2f756f3e4515a0b0bb5a
a504ebcdf8d9e8837b2d3b8b60eea5658be5dc9f3f9cddf1449b226d668591144a7bc4f17bf4e
51a56ea29b0f6d7321054f208e965b022a0668e2563a058626ea7347d9fb776e5596198fc99
1b1dd450b0e621ab9b11e10a0ab5bdb9f572f8f4b82f9edb5b17d154371118d3be51b28d085
42c2d7ff5e99674a070d0c08fcb55f8cfdc8a3739194b0d2ed99a34bbc70b0c43dc709be2bafc
e3255a6c747461b5bd24798160549d3ea9b37691039d4d5482640bcf3297cce75435867076
1f22876c78a4ac69ead

sign1=0x2be8a53416b161b0b7e76e28046a7a8b923417536fc6b27f2cb1666daaf6ab0292b6
96d9946134f3de1e4a9db1a0d47c5c888b0699bc29bf497dbcc49bd643bf151a06514d45f418
aca2198a6220d56970d7c15bc65caaac568148fa03f1d39d8cbf3b6e919e8ed3ff25655f15e18
```

```
e0b89109f8f337dcb80853dd22d73461dd3956e83e97debf6a5878f1219682cac4d2a71a2fc6
62743221d2faba4fe8ab2f02055bbc895c38475c95590c9b08beeec49217bbc52a377101202
2a9c537477892b1eeadf17ef5f9471d1d40abd097ca9026fa321df01add8289bf611bd4c029ec
ab9c0539a22b9360f241950bcb9b32dc339ca9c94054c0be4e1772629b7344a544c0eb5668
95d64ee773d220e417f9805bb36457515474146a22264c05c1a8f1a23c156c87111d4198dbe
e9e89fc14e059e0e9030c5d1ce2327ebbe99099d006c99e5d0cfb9ea2cd94b944c8933277e5
3f63a0e0664c9a652711ac0438bab41c6107924afaa77a5953c02e235a18f320822cdbb2621
ea0912631d55aefd1a4d

e=0x10001

N=0xe55c85be1e8f31b8cfa79da46e313545fe58d51308f427be1798373cce2304c0cee692f4
ab78387dc5d3161b5a1f33df90858c5c0a8fe906579257043a527f33e37b3466b7929be81abe
c6e9979215abf92d71032caf5fffe4a5f1c176172d8fb62da7beecc255e45b75a44e30ebbeb91
ecb97de7dc51a0c1d19f1cb0e5658b4a66cd4500252dc8f50076c357f5dece3f94ef1133cd2c5
92a5c9eb22a2e818f95252f0917caf47737807ece3a0f508f1af03b8eabd2f3d6cc881b27627e
3cb5eda7862c25213592ebf1f8470dff22d7603d299ee69628101c75133d65618692aad5f3b2f
fb3a22e1084a900cb0543107b02f8062737181eab4870cf25f0ed473cf4095530702314dd0a8
cace3a6fd0169f2dfea254d3ab152381c3ae535f780a1b532fe040eae7ba864bf28543a6dec71
1e62878ec4471341c8ee00824e9cae7627c29de36f3678cbfe046dce37bd6c7639c51f9387e1
b756bda7622efb9ee49fb258266b19fb359ef3f959ffabb0ebf3747bb923cf69899bfdcaba18bd
4dbb7

msg_ver1=power_mod(sign1,e,N)

msg1==msg_ver1
```

<u>Result for message/signature pair 1:</u>

```
True
```

Conclude: For message/signature pair 1, the verification returns True, therefore msg1 passes the verification.

The following command is input into sagemath for verification of **message/signature pair 2**:

<u>Commands:</u>

```
msg2=0x31c276bb243008ad8ee81ee029e80aad7e9ff16ed54dafe20649756f6bae1b57fb095
865c4a902d55892d0d22e3ba2ca62d7bc3a069a18f9c8df8e5b09a640ab1dd35bac240e6fbe
c27d7089abdcf943d3894cbe1e13a2db39dae0a1409259b76ead144ec7a8c308c4bd1ca1cc8
133de63c46d8092510ffa422bf8827d81e377bf70a07e6e82ed8b863cdc2aa6705731237f79f3
6aa6c35ca3f03542f0a7d5c56e3711b96b20c7bbb0da837c6cc3abec24783d2b95de9e6bc05
2b81d21955912d40a18b03ba9fcd206c37ccac389524b4ef4835822ea0cb3524ecd1a47ca2b
aa4bc66fd3dc4ba174aab59088019f9932102709519f8146d3d5af858077f351c97d277aaacf9
a832e0b4271475bf5fd29e380149e2bbd443f0e0363f7e96d2a3f02e384a01caf6b11afde5516
```

```
96f411f26e603225fde420deec3f4f715abb5e445180d2717870b2285f761f0baa91775151244
2ddbfd05e529a15b649f6d45aa93fb31626ab9a498d98612e225140c98551d0851057c33236
6e60f39234e0c711

sign2=0x79c8b72ca72f4c6363b3e29c1a6267ede2a1740dde90a071b8600f98f14eeee19b58
0213a872c00fb5146a851285945bda4728437622ab9e0f800881a7ebaf71cfcb558c8de0a150
e1452443808614f0e96dbbeedfabcdbf01e41b4b9601935bf9f12c5947d7a066c236d6843bcf0
5d136e1cd480eda39a40f3fa9e5a1b26033643859ad5b5bc91b185bd980d2efa223c5025c13
389e542167999282c8cb5aa180cfa89746f377bb3b2923bde3be1b6fa05980b6a80a9b52136
dd3b933dcd54a095dacb8d1c9fa7c8dbf96e421ae713440bba2f3c82c31c356a268d3623e7c1
510a8a6ca506943f843682c73179eaa35a9678ad599b2a41881a2b47234deb25640771b9ee
8ca4f488b21d735bf3adc1ae37a786dbc0622d7ba31a218d02567355af578b187eeef9de6b37
feb408f3ac296d5410c9d4d2920f452e30cf215227075756ff2f4fb0a15741102c31c1e5966276
7d78691bf864f3fbda722da50b8e02e88f7b6029eb85da47c44439b90e9cd4d027d171960b9
438d9c9c73d211e555

e=0x10001

N=0xe55c85be1e8f31b8cfa79da46e313545fe58d51308f427be1798373cce2304c0cee692f4
ab78387dc5d3161b5a1f33df90858c5c0a8fe906579257043a527f33e37b3466b7929be81abe
c6e9979215abf92d71032caf5fffe4a5f1c176172d8fb62da7beecc255e45b75a44e30ebbeb91
ecb97de7dc51a0c1d19f1cb0e5658b4a66cd4500252dc8f50076c357f5dece3f94ef1133cd2c5
92a5c9eb22a2e818f95252f0917caf47737807ece3a0f508f1af03b8eabd2f3d6cc881b27627e
3cb5eda7862c25213592ebf1f8470dff22d7603d299ee69628101c75133d65618692aad5f3b2f
fb3a22e1084a900cb0543107b02f8062737181eab4870cf25f0ed473cf4095530702314dd0a8
cace3a6fd0169f2dfea254d3ab152381c3ae535f780a1b532fe040eae7ba864bf28543a6dec71
1e62878ec4471341c8ee00824e9cae7627c29de36f3678cbfe046dce37bd6c7639c51f9387e1
b756bda7622efb9ee49fb258266b19fb359ef3f959ffabb0ebf3747bb923cf69899bfdcaba18bd
4dbb7

msg_ver2=power_mod(sign2,e,N)

msg2==msg_ver2
```

Result for message/signature pair 2:

```
False
```

Conclude: For message/signature pair 2, the verification returns False, therefore msg2 does not passes the verification.

(c) Attacker Marvin intercepted 2 signature/message pairs signed by Alice. Explain how Marvin can perform efficient forgery attack that given the 2 message/signature pairs and Alice's public key (N,e), compute a forgery message/signature pairs (msg3,sig3), where sig3 is a valid signature for new message msg3. Alice's private key, d, is not known to Marvin.

The following command is input into sagemath to show the efficient forgery attack that can be perform by Marvin and the verification on msg3/sig3 pair.

How was it done:

Algorithm used to produce digital signatures for messages such as RSA has multiplicative property, m' = m1*m2 , therefore, we can do m3 = (m1*m2)mod N to forge a valid message (m3). After that, we used the same method to forge another valid signature, sig3 = (sig1*sig2)mod N. Thus, the (msg3,sig3) pair is forged. At last, we compute another signature by using the forged message (msg3) and check if it equals to the forged signature (sig3). If the verification returns True, then the forged (msg3,sig3) pair is valid.

Commands:

```
msg1=0xa9edfdd28f7b79039fc041e8244d3d06bbac0e2cd108e1c5fd5691ae03545cf27a465
bee1216e6f97f5d2443767a32a6dcb46f012aa0ac19cf5e6b81097e2846cea4eab599620fc87
6ef6071d92b67cec573d91366301a60f90efab964ccfbda9b5fc197ba86bdfb9e7a5380f7e28c
90e7149a3d8ca5d443e22df5e284cbb5700c2f89df1c6a7d21dca87a856523bf1a37e44e85e7
6a03be641ff9366b6aa8552bee1763bca3cdba155899137fb8fae54fa4558a9cfcb5dfabcd147
3068b93ca

sig1=0x75b4bfc2420836f896bde21d195204a875867d43765194babbf67a3b9803515c7c177
9ecf9ab20266071493a3b2e12272e7ae1a1f030055c51eb1076814a7bdde56b9381644e892
a0a32d5b176af4428db1bddc53df9b28e0d7949c660559672fe497f051c978f407bbb961bfed8
841e5bd46d523010355535d01246da0d821a9389537e1747f8c296dff83ba22bbd5993300c9
2846ea288aade9fb0591c3bb3dd25372c15224a5ac3588734144190fce710a2e07493cc6bb0
ad80f205667a4264ab0d1b139b8ac8fc2f35a89ec2f9c6c159f683ef111796eb8d6b0d02d73f5
cc890af958c9ea7fc9ada016bb53c80191babb37facd0ead5093a969cc5947775f48b5b80797
533d1cd40987eb537d63221c51dc87863d5ba6fe30fea2d1831fda5334e16b6e2c6db117d7e
b5bcc918f6718213f6902fce55b498ca1f381fba98f49c6858c65168de2416cb8408f6838d12e
5c99c6d1e0aea0e6c3dcbda5075dea991ac42c759b835e9bd4ff303eab6702e798ed284ebed
a6e50754828cad511b

msg2=0x4eaacda480337afeea561e82087deb98a9b16e84a7fdb9f6c586e37e04f74c42a891
d2dbb397154a0e76b7df72c0975702965a1ca2a70bad04a7285b0a5618ef9bca9070f76c193
0225f56f58d17b15b8954a35e08223f505c9c2e93cf8bf7f28c50647d385863dc36bd9e52556c
896e89e9073015a7c38a59f29141247014518444450a2e4c792b70e99bc730f82005154f7e6c
79b4ef394aaa155c524c18da44ea8615ca0cce07aa822c0ec6704902bef72db21f3302cfccf3a
b79458c1f197

sig2=0x77dd1b15bcf1cf5dc65d64408a18771a7cfb2af1833de20d7802ddd85d2c09429f52bf9
e1c2700a39f22f2d98933e22fc57876f30a1d86696a6970fae76d89e5556046fe1f91a0701571
4fc3d302e326bbe364f5784ffb0a6c68f33a693a758e442e1e2410731d2f6e77776e0435f5d80
```

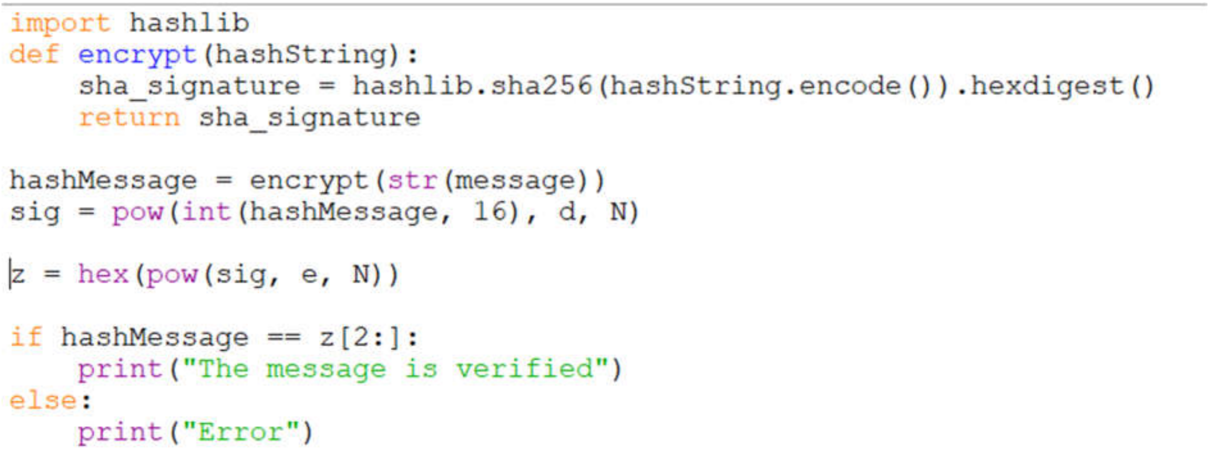
```
6c8b149aef99fcc8f49620b487a6703178839d3658b4c5008e2db841383a9aceb0862aec852f
6af50926f9045afff5e40ab52929993e3b79bca23777f06ea2914a284b90254798d131dfd0e8b
97d0c94bfa589ed9a75a4d4ea045fecf7325eb6e91ca6536113d6acbe6e83a1dfac9833edafb
8d2d25439a8eb0ad6cec289d8628cc0733ca231409065c3215e3ad5e2f5ad61756e6d4bd5b
14fffb78e9d1f470057061f0b9ab988f3be141435a8a71f70f7084a20f6ce13ac2aceba4546afc6
fd28b305dbde81c7b54be192c69f118ce5d131b5f217e2ec804c1e11578cbff51270a4fd16632
2dd69aa48a18466f608c2621baf828002d55c8361cbb1f7b3279c986bdf1c3da2c12fe565e7c
15d7250

d=0x541af9701e04a45700ce962015c835a0d503fe1e5cca2b48a99e47a32473f2ea40f48c2e
ec31c98555657255d5565bcf3f4fb98886d6febc34a0950817dae88a3e808f569b3a47b1751d
4013a861095166ae2322e6dfe8740d844c8284ab3b29d7c4261efcf2c64c56bd6ce2bf4db342
6ee879683cf669f6c7351c55398cb03a8e4c9a0e3ccbe5d527a3912a8cea045414b7bdef2ffe9
a348c56dec274ba676e05a224553543910fe6940169f73be36bbca1c0cd53525f53e4b2aa9e
69423ef077b2d1bfe8d45927a677f74418240b95ed5c698e62fb429ece5fabbdcff8f64c480bff4
6bb6a448ae350739795abd156a5814378248b7100bfbaa07b039bc105a32a6fe74e0768857
7edecd515bd452a41cbfc017b9d26e76a5bec2ce433714a02f0f2c3784b65738adc849c3c31f
8a731132e4bd8c2b2c0b33de87403c2b7ff12ab3d9582453844b4ff03142f899b256e407c330
1adc46794d14bd668beac877e9cb5aa0602c447b75d3424d3a71495ed55a86fb1b01b5fbae2
a766f6172301

N=0xe55c85be1e8f31b8cfa79da46e313545fe58d51308f427be1798373cce2304c0cee692f4
ab78387dc5d3161b5a1f33df90858c5c0a8fe906579257043a527f33e37b3466b7929be81abe
c6e9979215abf92d71032caf5fffe4a5f1c176172d8fb62da7beecc255e45b75a44e30ebbeb91
ecb97de7dc51a0c1d19f1cb0e5658b4a66cd4500252dc8f50076c357f5dece3f94ef1133cd2c5
92a5c9eb22a2e818f95252f0917caf47737807ece3a0f508f1af03b8eabd2f3d6cc881b27627e
3cb5eda7862c25213592ebf1f8470dff22d7603d299ee69628101c75133d65618692aad5f3b2f
fb3a22e1084a900cb0543107b02f8062737181eab4870cf25f0ed473cf4095530702314dd0a8
cace3a6fd0169f2dfea254d3ab152381c3ae535f780a1b532fe040eae7ba864bf28543a6dec71
1e62878ec4471341c8ee00824e9cae7627c29de36f3678cbfe046dce37bd6c7639c51f9387e1
b756bda7622efb9ee49fb258266b19fb359ef3f959ffabb0ebf3747bb923cf69899bfdcaba18bd
4dbb7

msg3=Mod((msg1*msg2),N)

sig3=Mod((sig1*sig2),N)

check=Mod((msg3^d),N)

print(check==sig3)
```

Result:

```
_True
```

(d) How Alice can modify the digital signature scheme to prevent forgery attack in (c) without changing the keys from (a). Show the new signature of the same message from (a) signed by her private key (d,N) using the modified signature scheme.

The code below was done in Python program to assure the message is the same with the decrypted signature. Hash library has to be imported before the function as in the first line. The message has to be converted to UTC-8 character coding then hexadecimal before entering into the hash function. Message has to be hashed using SHA256 before encrypting with the public key.



```
import hashlib
def encrypt(hashString):
    sha_signature = hashlib.sha256(hashString.encode()).hexdigest()
    return sha_signature

hashMessage = encrypt(str(message))
sig = pow(int(hashMessage, 16), d, N)

z = hex(pow(sig, e, N))

if hashMessage == z[2:]:
    print("The message is verified")
else:
    print("Error")
```

In this case, Alice's message has to be hash using SHA256 before encrypting with Bob's private key. Signature received by Bob then decrypt by his private key. the message is then compare with the signature. If they are the same means the message is verified, else the message is not verified.

2 Task 2

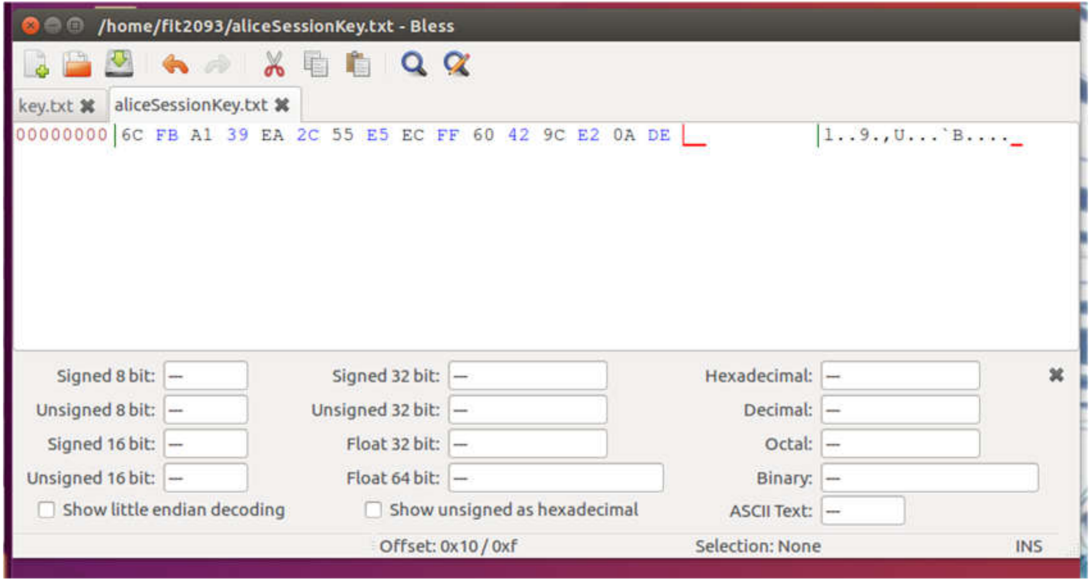
(a)use “Hybrid” encryption method combining RSA and AES-128 in 8-bit CFB mode to send an encrypted file to Bob.

Given:
Bob’s public key file, bob.pub
Alice’s AES-128 session key, K=0x6cfba139ea2c55e5ecff60429ce20ade,
CFB-8 IV value, IV=0x6b66a2ce1972853f84d1874736369036,
plaintext file, msg qa.bin

Steps:

First, we will create a txt file that would contain Alice’s session key by using Bless tool in Unix.

- we paste the K into Bless and save it as aliceSessionKey.txt for further usage.



/home/fit2093/aliceSessionKey.txt - Bless

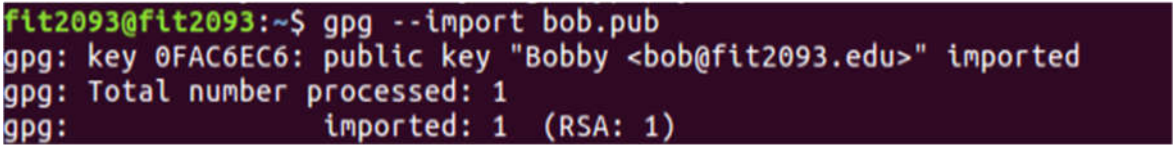
```
00000000 6C FB A1 39 EA 2C 55 E5 EC FF 60 42 9C E2 0A DE     1..9.,U...`B....
```

```
  Signed 8 bit: —        Signed 32 bit: —        Hexadecimal: —
Unsigned 8 bit: —      Unsigned 32 bit: —            Decimal: —
 Signed 16 bit: —          Float 32 bit: —              Octal: —
Unsigned 16 bit: —         Float 64 bit: —             Binary: —
□ Show little endian decoding   □ Show unsigned as hexadecimal   ASCII Text: —
                         Offset: 0x10 / 0xf        Selection: None        INS
```

Then, the saved session key file is encrypted by using the RSA algorithm.

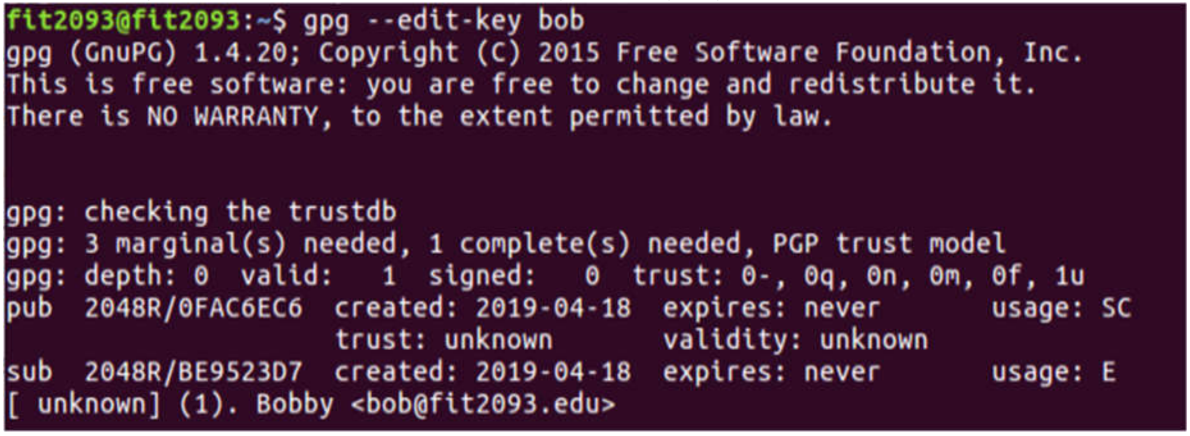
Commands:

# first, we import Bob’s public key into the system.

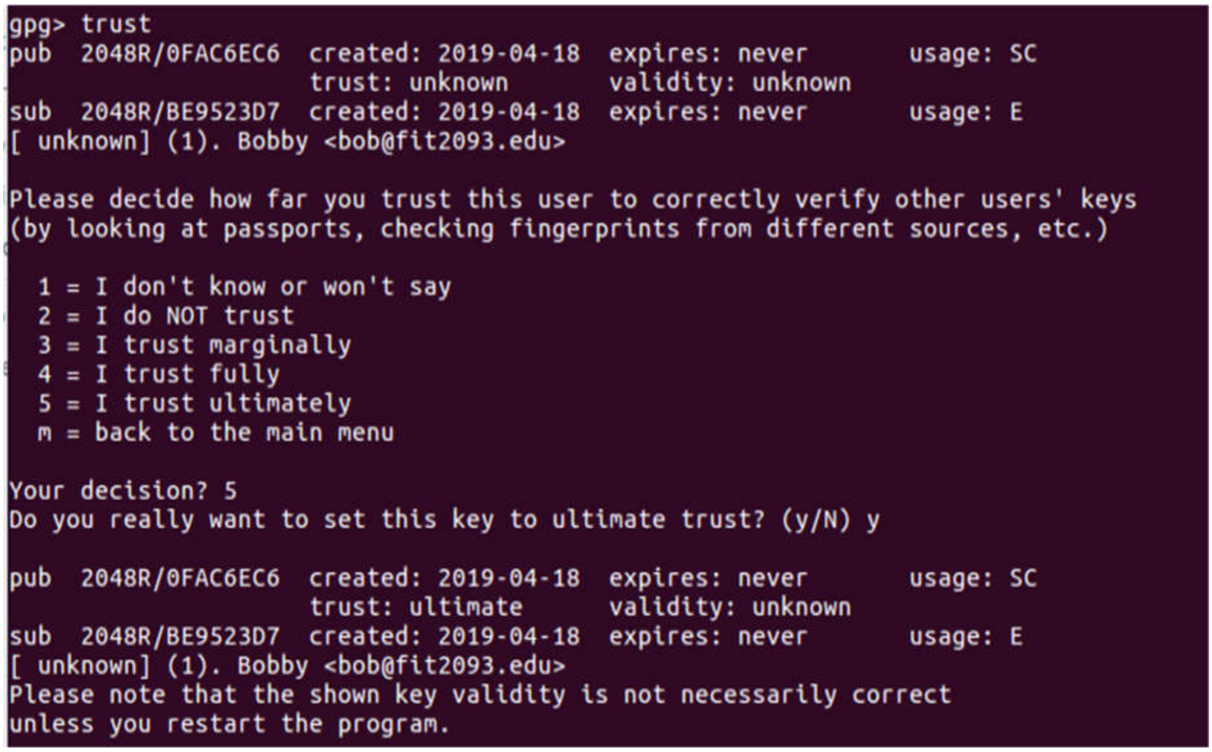


```
fit2093@fit2093:~$ gpg --import bob.pub
gpg: key 0FAC6EC6: public key "Bobby <bob@fit2093.edu>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

# then we update the trust to ultimate for the public key imported.

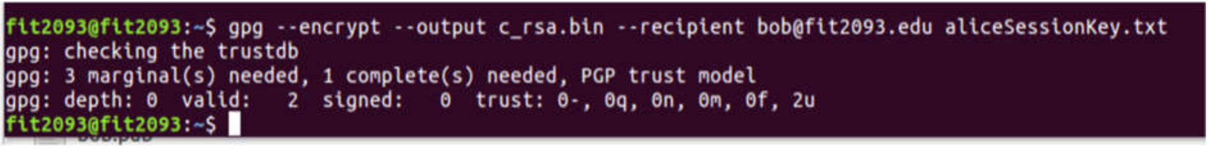


```
fit2093@fit2093:~$ gpg --edit-key bob
gpg (GnuPG) 1.4.20; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.


gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub  2048R/0FAC6EC6  created: 2019-04-18  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  2048R/BE9523D7  created: 2019-04-18  expires: never       usage: E
[ unknown] (1). Bobby <bob@fit2093.edu>
```



```
gpg> trust
pub  2048R/0FAC6EC6  created: 2019-04-18  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  2048R/BE9523D7  created: 2019-04-18  expires: never       usage: E
[ unknown] (1). Bobby <bob@fit2093.edu>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y

pub  2048R/0FAC6EC6  created: 2019-04-18  expires: never       usage: SC
                     trust: ultimate      validity: unknown
sub  2048R/BE9523D7  created: 2019-04-18  expires: never       usage: E
[ unknown] (1). Bobby <bob@fit2093.edu>
Please note that the shown key validity is not necessarily correct
unless you restart the program.
```

# encrypt the session key with RSA algorithm



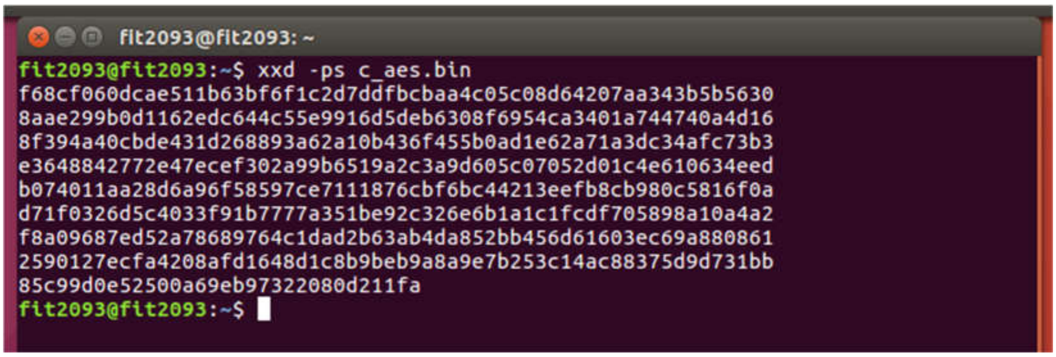
```
fit2093@fit2093:~$ gpg --encrypt --output c_rsa.bin --recipient bob@fit2093.edu aliceSessionKey.txt
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
fit2093@fit2093:~$
```

Then we proceed to the encryption of the plain text file by using AES-128-CFB8 algorithm.

Commands:

~$ openssl enc -aes-128-cfb8 -in msg_qa.bin -out c_aes.bin -K 6cfba139ea2c55e5ecff60429ce20ade -iv 6b66a2ce1972853f84d1874736369036 -nosalt

the output c_aes.bin in hexadecimal form:



```
fit2093@fit2093:~$ xxd -ps c_aes.bin
f68cf060dcae511b63bf6f1c2d7ddfbcbaa4c05c08d64207aa343b5b5630
8aae299b0d1162edc644c55e9916d5deb6308f6954ca3401a744740a4d16
8f394a40cbde431d268893a62a10b436f455b0ad1e62a71a3dc34afc73b3
e3648842772e47ecef302a99b6519a2c3a9d605c07052d01c4e610634eed
b074011aa28d6a96f58597ce7111876cbf6bc44213eefb8cb980c5816f0a
d71f0326d5c4033f91b7777a351be92c326e6b1a1c1fcdf705898a10a4a2
f8a09687ed52a78689764c1dad2b63ab4da852bb456d61603ec69a880861
2590127ecfa4208afd1648d1c8b9beb9a8a9e7b253c14ac88375d9d731bb
85c99d0e52500a69eb97322080d211fa
fit2093@fit2093:~$
```

for your convenience, below is the output in text format:
0xf68cf060dcae511b63bf6f1c2d7ddfbcbaa4c05c08d64207aa343b5b5630
8aae299b0d1162edc644c55e9916d5deb6308f6954ca3401a744740a4d16

```
8f394a40cbde431d268893a62a10b436f455b0ad1e62a71a3dc34afc73b3
e3648842772e47ecef302a99b6519a2c3a9d605c07052d01c4e610634eed
b074011aa28d6a96f58597ce7111876cbf6bc44213eefb8cb980c5816f0a
d71f0326d5c4033f91b7777a351be92c326e6b1a1c1fcdf705898a10a4a2
f8a09687ed52a78689764c1dad2b63ab4da852bb456d61603ec69a880861
2590127ecfa4208afd1648d1c8b9beb9a8a9e7b253c14ac88375d9d731bb
85c99d0e52500a69eb97322080d211fa
```

(b) Show Bob's decryption process and the decrypted plaintext.

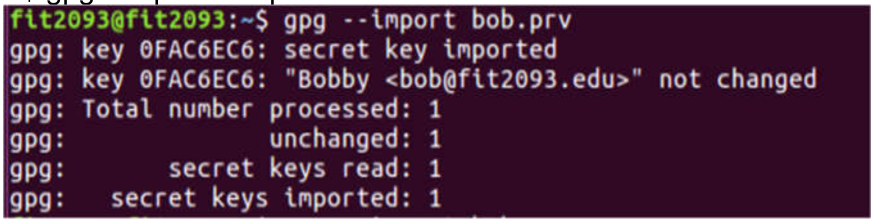
Given:
Bob's private key, bob.prv
IV used, iv=0x5fe4bbaf52dfd660407a9a8e123901ea
Bob's gpg passphrase: fit2093

Steps:

First, we have to import the Bob's private key into the system.

Commands:

```
~$ gpg --import bob.prv
```



```
fit2093@fit2093:~$ gpg --import bob.prv
gpg: key 0FAC6EC6: secret key imported
gpg: key 0FAC6EC6: "Bobby <bob@fit2093.edu>" not changed
gpg: Total number processed: 1
gpg:               unchanged: 1
gpg:        secret keys read: 1
gpg:    secret keys imported: 1
```

Then, we have to decrypt the session key from c_rsa_qb.bin that was encrypted with RSA algorithm.

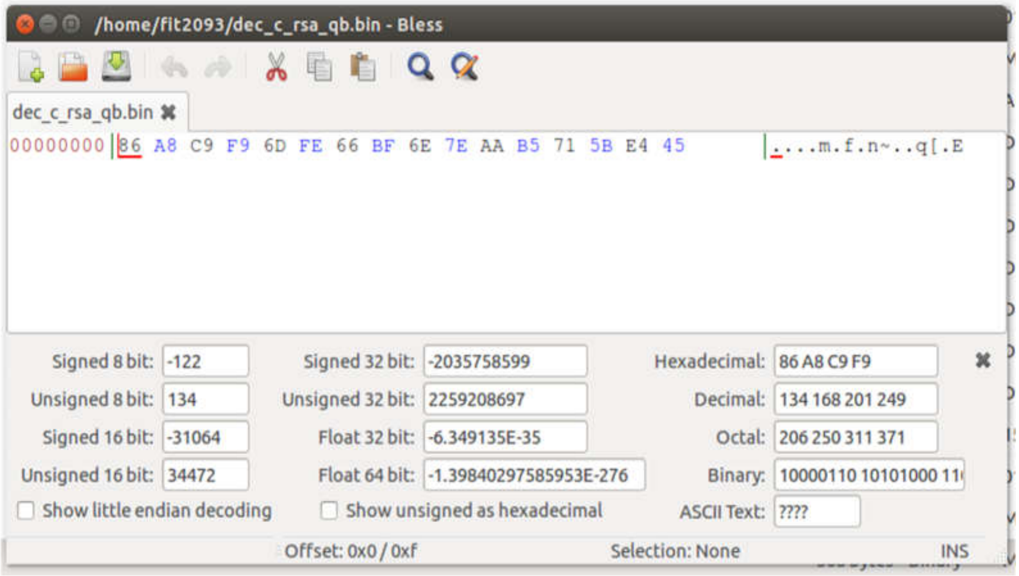
Commands:

```
~$ gpg --output dec_c_rsa_qb.bin --decrypt c_rsa_qb.bin

You need a passphrase to unlock the secret key for
user: “Bobby <bob@fit2093.edu>”
2048-bit RSA key, ID BE99523D7, created 2019-04-18 (main key ID 0FAC6EC6)
```

# after entering the correct passphrase (fit2093), the secret key is successfully extracted from c_rsa_qb.bin file as saved as dec_c_rsa_qb.bin .

```
gpg: encrypted with 2048-bit RSA key, ID BE9523D7, created 2019-04-18
      “Bobby <bob@fit2093.edu>”
```

The decrypted session key file is saved as a binary file (dec_c_rsa_qb.bin). Therefore, we use a hex editor (Bless) to get the session key in hexadecimal form.



/home/fit2093/dec_c_rsa_qb.bin - Bless

dec_c_rsa_qb.bin ✖

```
00000000 86 A8 C9 F9 6D FE 66 BF 6E 7E AA B5 71 5B E4 45    ....m.f.n~..q[.E
```

Signed 8 bit: -122

Unsigned 8 bit: 134

Signed 16 bit: -31064

Unsigned 16 bit: 34472

□ Show little endian decoding

Signed 32 bit: -2035758599

Unsigned 32 bit: 2259208697

Float 32 bit: -6.349135E-35

Float 64 bit: -1.39840297585953E-276

□ Show unsigned as hexadecimal

Hexadecimal: 86 A8 C9 F9

Decimal: 134 168 201 249

Octal: 206 250 311 371

Binary: 10000110 10101000 11

ASCII Text: ????

Offset: 0x0 / 0xf   Selection: None   INS

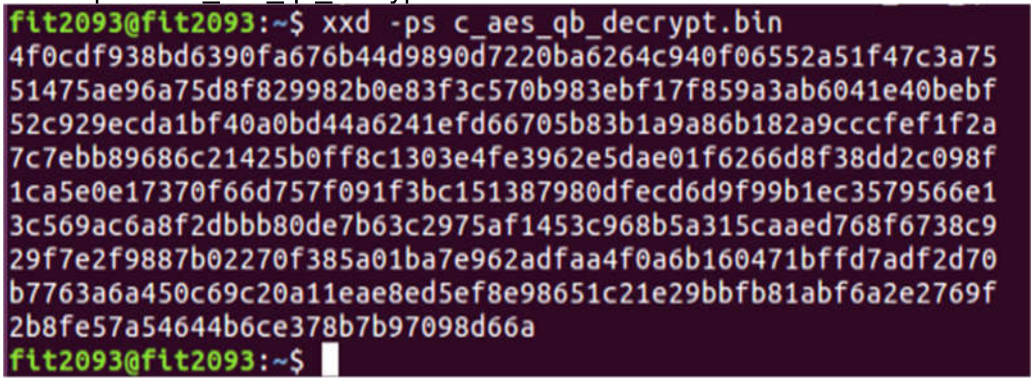
As shown above in the picture, the session key is,
K=0x86a8c9f96dfe66bf6e7eaab5715be445

Now, we use the session key and the IV given to decrypt the message c_aes_qb.bin.

Command:

~$ openssl enc -d -aes-128-cfb8 -in c_aes_qb.bin -out c_aes_qb_decrypt.bin -K 86a8c9f96dfe66bf6e7eaab5715be445 -iv 5fe4bbaf52dfd660407a9a8e123901ea -nosalt

the output file c_aes_qb_decrypt.bin in hexadecimal:



```
fit2093@fit2093:~$ xxd -ps c_aes_qb_decrypt.bin
4f0cdf938bd6390fa676b44d9890d7220ba6264c940f06552a51f47c3a75
51475ae96a75d8f829982b0e83f3c570b983ebf17f859a3ab6041e40bebf
52c929ecda1bf40a0bd44a6241efd66705b83b1a9a86b182a9cccfef1f2a
7c7ebb89686c21425b0ff8c1303e4fe3962e5dae01f6266d8f38dd2c098f
1ca5e0e17370f66d757f091f3bc151387980dfecd6d9f99b1ec3579566e1
3c569ac6a8f2dbbb80de7b63c2975af1453c968b5a315caaed768f6738c9
29f7e2f9887b02270f385a01ba7e962adfaa4f0a6b160471bffd7adf2d70
b7763a6a450c69c20a11eae8ed5ef8e98651c21e29bbfb81abf6a2e2769f
2b8fe57a54644b6ce378b7b97098d66a
fit2093@fit2093:~$
```

for your convenience, below is the output in text format:
0x4f0cdf938bd6390fa676b44d9890d7220ba6264c940f06552a51f47c3a7551475ae96a75d8f
829982b0e83f3c570b983ebf17f859a3ab6041e40bebf52c929ecda1bf40a0bd44a6241efd667
05b83b1a9a86b182a9cccfef1f2a7c7ebb89686c21425b0ff8c1303e4fe3962e5dae01f6266d8f
38dd2c098f1ca5e0e17370f66d757f091f3bc151387980dfecd6d9f99b1ec3579566e13c569ac
6a8f2dbbb80de7b63c2975af1453c968b5a315caaed768f6738c929f7e2f9887b02270f385a01
ba7e962adfaa4f0a6b160471bffd7adf2d70b7763a6a450c69c20a11eae8ed5ef8e98651c21e2
9bbfb81abf6a2e2769f2b8fe57a54644b6ce378b7b97098d66a

(c)The mathematical formula to calculate the number of blocks which are corrupted is:

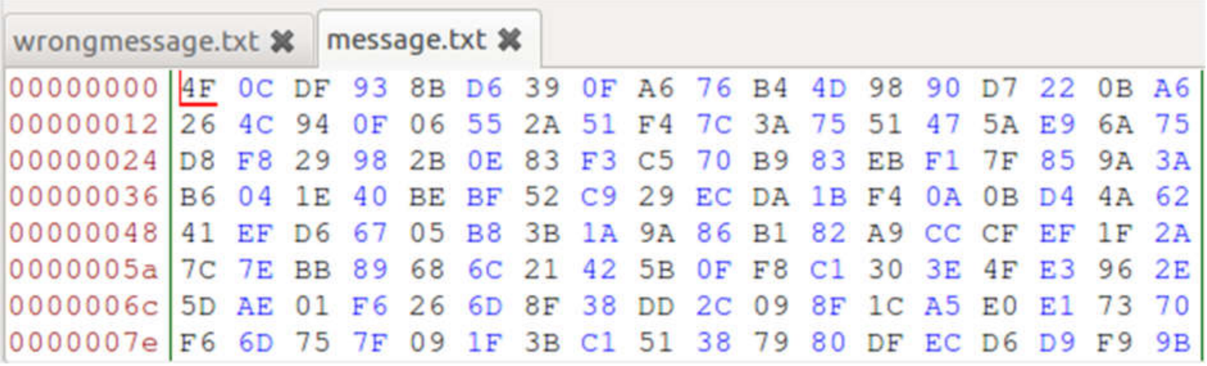
(total number of bits from aes / stream size of cfb) +1

Number of blocks corrupted can be calculated by the formula above. The total number of bits from AES is 128 and the stream size of cfb is 8 bits in this case. Therefore, we can substitute 128 and 8 into the formula respectively as the output of AES will split into streams of 8 bits in the cfb cipher. One has to be added to account for the plaintext which has error. This will result in block size / the amount of bits shifted block size of output being garbled.

From this question, there is an error in second block, incorrect plaintext will output by the cipher until the shift register once again equals a state it held while encrypting, at which point the cipher has resynchronized in nineteenth block. This will result in block size / the amount of bits shifted block size of output being garbled.

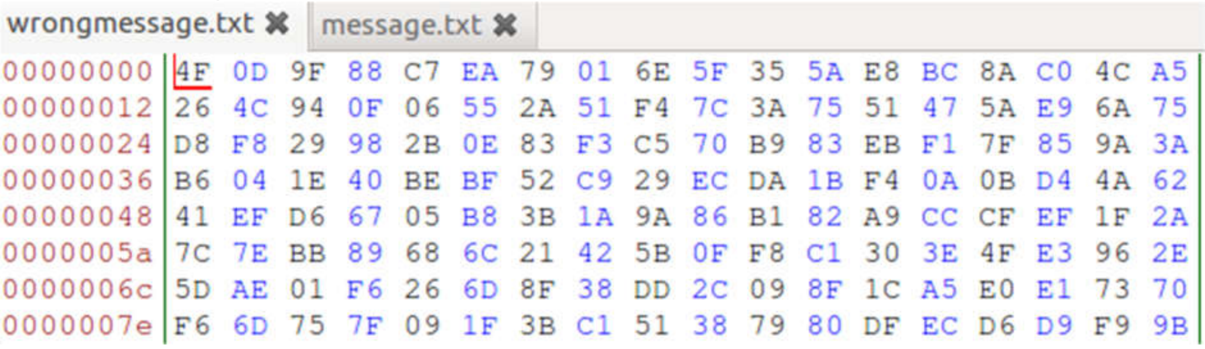
after c aes qc.bin has decrypted, file wrongmessage.txt is open for display the message in hex using hex editor.

from file < c aes qb.bin> :



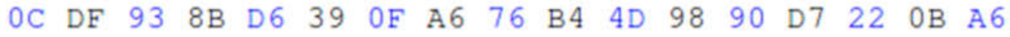
```
00000000 4F 0C DF 93 8B D6 39 0F A6 76 B4 4D 98 90 D7 22 0B A6
00000012 26 4C 94 0F 06 55 2A 51 F4 7C 3A 75 51 47 5A E9 6A 75
00000024 D8 F8 29 98 2B 0E 83 F3 C5 70 B9 83 EB F1 7F 85 9A 3A
00000036 B6 04 1E 40 BE BF 52 C9 29 EC DA 1B F4 0A 0B D4 4A 62
00000048 41 EF D6 67 05 B8 3B 1A 9A 86 B1 82 A9 CC CF EF 1F 2A
0000005a 7C 7E BB 89 68 6C 21 42 5B 0F F8 C1 30 3E 4F E3 96 2E
0000006c 5D AE 01 F6 26 6D 8F 38 DD 2C 09 8F 1C A5 E0 E1 73 70
0000007e F6 6D 75 7F 09 1F 3B C1 51 38 79 80 DF EC D6 D9 F9 9B
```

from file <c aes qc.bin>:



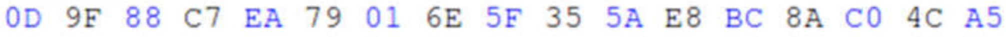
```
00000000 4F 0D 9F 88 C7 EA 79 01 6E 5F 35 5A E8 BC 8A C0 4C A5
00000012 26 4C 94 0F 06 55 2A 51 F4 7C 3A 75 51 47 5A E9 6A 75
00000024 D8 F8 29 98 2B 0E 83 F3 C5 70 B9 83 EB F1 7F 85 9A 3A
00000036 B6 04 1E 40 BE BF 52 C9 29 EC DA 1B F4 0A 0B D4 4A 62
00000048 41 EF D6 67 05 B8 3B 1A 9A 86 B1 82 A9 CC CF EF 1F 2A
0000005a 7C 7E BB 89 68 6C 21 42 5B 0F F8 C1 30 3E 4F E3 96 2E
0000006c 5D AE 01 F6 26 6D 8F 38 DD 2C 09 8F 1C A5 E0 E1 73 70
0000007e F6 6D 75 7F 09 1F 3B C1 51 38 79 80 DF EC D6 D9 F9 9B
```

the correct plaintext blocks:



```
0C DF 93 8B D6 39 0F A6 76 B4 4D 98 90 D7 22 0B A6
```

the wrong plaintext blocks:



```
0D 9F 88 C7 EA 79 01 6E 5F 35 5A E8 BC 8A C0 4C A5
```

There is an error in second block of file <c aes qc.bin> which cause the plaintext blocks differ with the message in the first row of <c aes qb.bin>. Seventeen plaintext blocks are affected. The plaintext block corresponding to the ciphertext block is obviously altered. In addition, the altered ciphertext block enters the shift register and is not removed until the next sixteen blocks are processed.

(d) The attacker obtained 2 ciphertext sent by Alice: c_aes1.qd.bin, c_aes2_qd.bin & msg1_qd.bin
Alice use SAME K and IV in encryption software.

<u>How it was done:</u>

First, we know that by the encryption rule for one-time pads, where **K is the reused pad(use same key)**:

$$p_1 \oplus k = c_1 \quad \& \quad p_2 \oplus k = c_2$$

by XOR the two ciphertext, we can get that:

$$c_1 \oplus c_2 = (p_1 \oplus k) \oplus (p_2 \oplus k)$$

by rearranging the equation:

$$c_1 \oplus c_2 = (p_1 \oplus p_2) \oplus (k \oplus k)$$

we know that $a \oplus a = 0$ for all a,this is because everything is its own inverse. Therefore,

$$c_1 \oplus c_2 = (p_1 \oplus p_2) \oplus 0$$

$$c_1 \oplus c_2 = (p_1 \oplus p_2)$$

As the attacker obtained 2 ciphertext sent by Alice, we first compute the XOR of the 2 ciphertext, let's call it x:

$$x = c_1 \oplus c_2$$

After we have obtained x, we can say that:

$$x = p_1 \oplus p_2 \quad \text{, because } c_1 \oplus c_2 = (p_1 \oplus p_2)$$

And if we XOR the obtained p1 with x:

$$x \oplus p_1 = (p_1 \oplus p_2) \oplus p_1$$

we rearrange the equation, and we can get that:

$x \oplus p_1 = p_2 \oplus (p_1 \oplus p_1)$ and we know that $a \oplus a = 0$

therefore,

$$x \oplus p_1 = p_2 \oplus 0 = p_2$$

At the end, we can acquire the plaintext that correspond to c2.

<u>Commands:</u> (The followings codes runs in SageMath)

```
c1=0x9645fa574f9a09991cdc70d228f7df0f8a08a8f761085819af5f0a2d6c5021ae11e893e7c
95932af3e3a45923958da24669fcacf1857308d176ea760fb321fc21b2618881d2809c15ff985
249156a0dd1f2e09c781914de8ac589fdf8f93a204540ff593df49287849ac7242691f6400a87f
a62750a60e0c303c6b6b363b30d3efb13836e680ab74f92302ff2b66beabd0310804089783e
1ad015403c7c2d52dfb983fa4469613e672ba49bd6a3cf1a433578b4d144a93638f3e5e1ad7d
95c918aa02bd93352f5f6f846158b3789c4b16015ac829d49c9d4ee7f2520b4474124eb0b414
89bcc2d04d1a6983f03a2aea31361d4fa1efb4514edfa1ec2b670d5b0

c2=0xb6d0ab7deef3c89fe7ba2e3adb070c383c79ecc9cc2de4c794631cecc16c9428de862fc
9da375bae0751947aac4fbb011691195e7a1ec86603632ecc7477c35236e9c971d9fe1f1c4f4
3f461a011966441b416a1e7cf5ded5b72d6ad2e02dc1f007a25f80133e21484f1f4d66c9fb6be
51ccc26ed660ec9f2fe58463a50e611be63eb4e1f629ce382a584588015bf0f0b0b6ea7f7a5bd
```

```
e4b347dd70cb214b64ab42f8766a20ca0ec8a1d274a30a207928f0447077f3b4fe70b6f4d48b
71ec28d017e2dec3ef068ed7da4db71b3f14cdf096e3260a6474338dcbd6523fd2e61164781b
232616615b5a9835dbd14d421ab90f8e700d922f9f4b73a7dba9f59a8ec

p1=0xfc6c7477b2a0d191057c3806d7ff7fad64f610c42575e793fc8e16a6151825d9af9987c5
1f624d884d86518989107a95b279a8e4e16e72cbcc0d1efe230142dbf7733a52bad70ce6723
18eeedbf289723720cc05f4b4deb7fa93654348529fa499a72f0ce1ffa3b6c0f819814c7500b9a
161f0a712ba153ae9baa557ccb825ed9bc3677d95e37dc665e4e6491c69fe53ade60bf3ee7e
e8b2b3549cc89a6e03b02f766c24f47ac01e502d936e35a075c67cbd84c2719909886170f7fc
d3430fd5ec5cf024fad7f32a8e1cfa05b3909410a0c276f1d9cfc71381cebb97293d937c80dc4e
6c35d20915b73e3941f6be678a7918485af1ae57cc9db0c419691e8ea0

# x is the XOR of the two obtained ciphertext
x=c1^^c2
# y is the XOR of the obtained plaintext and the XOR of the two obtained ciphertext
y=x^^p1

print(hex(y))
```

Result:

```
0xdcf9255d13c91097fe1a66ee240fac9ad28754fa88505b4dc7b20067b824905f60f73beb0c0
c248974ed80611c071bb0c2777b7583278a20d8009752ac449e4bdabcebab7e011a3b628bff
abeab5bfcb69bad36392eaceb20db92c31e9c3e1bfcdd2ff673f8569da0da59f1549f5d20758d
294ee947cf7a9f6634a5f5f8d7425924cebaa854a188ab69fa13e3654b008cd61e9889cb2b51
82a281fc7efb860d760c1d4e610e07314a88afd996f3e83f0c0ee48881ae8d50ce521e4aeb38
491c96782f611f7756e310bfe34ff37e81c7ec9b9e813e2c14dffb30cfb946054b34e2c56bd16c
f7831a4cf1bfcc3e1c8e882fa817ba03677eb2cc770a7614037f3fc
```

Task 3

(a)Explain an efficient attack against the above mitigation technique. Given the public modulus N, Alice's public key e = 10001 in hexadecimal, and the ciphertext c' encrypted by using the above mitigation technique, show how the attacker Marvin can recover the PIN efficiently without Alice's private key.

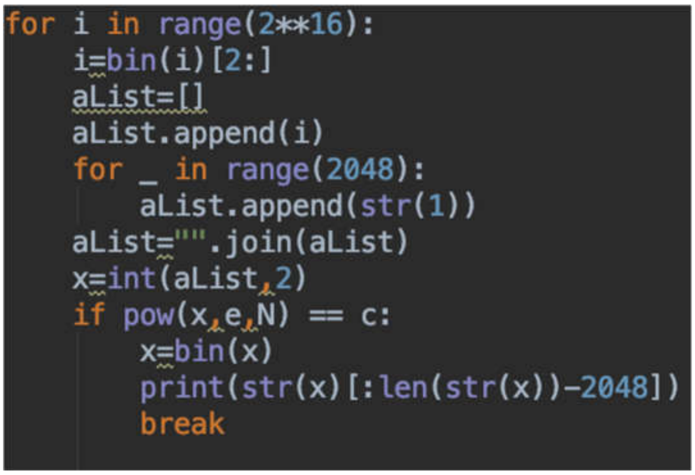
How it was done:

Marvin can use brute force to recover the pin as the PIN is a positive integer smaller than 2^{16}therefore it can be run in a reasonable time to try and find the PIN.
For every integer smaller than 2^{16}, we convert it to binary then append 2048 "1" at the rightmost side. Then by encrypting using $x^{e} mod\ N$we get a ciphertext then compare it to c_prime.
Once it matches c_prime, from x we remove the last 2048 "1" and we can obtain its PIN.

Steps:

N=0xa05bb3783769b832a2d022646c48344948282cdcd42bff414ec90f23b7e3b1b817137664f40163
19586395741996245f9c2c66dee453352dc329fe54228beaa559a610114dbe902c32572e954660adbd
06f8da8c770c33bb5ad15f506073ea0c50ff4e9906e16ee70d1311e0ad81896f4807282361f5b211648
8de06966b571cdb15da536226378bc1fba8a3476c5809b5a274a0117b5de3e52278d39fdfa62de29f3
38b0453ac3af61a30dcb2975949a3d0ec2d2b7f0d2c4d2e3ef6ddefa8caad21bc16972dcecfcd5f93323
73a759632f7f02c52dd424b83985eaa673ce67023366e85899729fc1d1fede02fa9c53aa01328c9108a3
c5145f47ef988688f3076d49821314210d1f4db88fa836d41f3dc3960499eb46b28261aaa1515e0fb6d
7481ae051b607683cbfdc18d6b692f93d6facf4002d6fa835aac4d61911b66859a81043763e1d0ef6e47
f1a7a4c8d57993b0fb67b5758ed3aca9540d39e150935cdd0c320d166da65612ae78322f96853885e6
a44add306a899fab2f87cf2a1d

c_prime=0x90fdea0c662ed2cef739c491c2f391d8cf636b80144c412580c02e3262e4fa10c6e101f4c53
d09619c7cb6fc9d8edfe2a676c1c128bd8e32528aff243101b9daf655bcd5460a9bf020ff4bef0f61b943
04b142b6b18830b8b4d5574e8b54903de67df71f39234fdf9f66723ab1bf426d1c0a95fabae8485e9edf
7f4c868ca2816398b1f46ffda2a84b5d52ff36bad829ddc2e123f86cb266256824f047fb6f6a1c7593eaf4
ae5c47c6f5e633370d832345fde53324d02687a9b21e60fcefb5e2e2eb1ace969fe72afca67847acad09
3dec8976336ace5f135257f740f625851a3258854775a3f4f123eae1a6253b2740de37d112bca596f36e
4c0d4cfc50b05643b8ec0b52619ae7d0ae990e041ba01bc149ac4a510c81e3aef3f4f2843a50f15c637e
274e714c6a768e0c7d96e28a5365b64aee0315623794724573648516ebc9b0f5135a180ac3141a98f2
ef0f005f6980781036c9b1c7975774708d1929d1935ae782de80722124220a9dd3fadc457d8bdb8be7
62b0158187ee619142637d



```
for i in range(2**16):
    i=bin(i)[2:]
    aList=[]
    aList.append(i)
    for _ in range(2048):
        aList.append(str(1))
    aList="".join(aList)
    x=int(aList,2)
    if pow(x,e,N) == c:
        x=bin(x)
        print(str(x)[:len(str(x))-2048])
        break
```

Results:

The PIN in binary: 0b1100011100000111(50951)

(b) An implementation of the RSA-3072 encryption software that Alice uses has the following vulnerability: the software neglects to clear the value of ϕ(N) from the memory after the key generation process. An attacker Marvin who manages to gain access to Alice's computer exploits this vulnerability by performing a memory dump of the machine after Alice completed her key generation, to get the value of Alice's ϕ(N). Explain how Marvin can efficiently factorise N by using this additional information. Show how to factorise the modulus N from (a) by using the following ϕ(N).

How it was done:

When ϕ(N) is given and N = *pq* where *p* and *q* are prime numbers, then we have

$$\varphi(N) = (p-1)(q-1) = pq - (p+q) + 1$$

But *pq* = n, therefore ,

$$\varphi(N) = N - (p+q) + 1 \; and \; p + q = N + 1 - \varphi(N)$$

Now, *p* and *q* are the roots of the equation,

$$x^2 - (p+q)x + pq = (x-p)(x-q)$$

Substituting for *p+q,*

$$x^2 - (N + 1 - \varphi(N))x + n = (x-p)(x-q)$$

By using sagemath tool:

```
phi=0xa05bb3783769b832a2d022646c48344948282cdcd42bff414ec90f23b7e3b1b8171376
64f4016319586395741996245f9c2c66dee453352dc329fe54228beaa559a610114dbe902c3
2572e954660adbd06f8da8c770c33bb5ad15f506073ea0c50ff4e9906e16ee70d1311e0ad818
96f4807282361f5b2116488de06966b571cdb15da536226378bc1fba8a3476c5809b5a274a0
117b5de3e52278d39fdfa62de29f338b0453ac3af61a30dcb2975949a3d0ec2d2b7f0d2c4d2e
3ef6ddefa8c915dbdc153c25b847c313f96c30a78950106adcc70eef014d1340f26f0fd36a90d
6a5e1c369a70658dfbb20feccf4efd255d477924a95ae093387182cf946b4dae80d6b434fcb11
f2a8e9265b23e7dd076733f268d8cacf0ba15aeee50b7fc577c7db1269f54436c8c9ade14d23f
27097e128a4a312eb9c9e7cd9fc5c40efe18a6b3b56947761243265d6a3ccd7bc9027e6e4ec
c267765ea293574502e955349b0d866ad5a16f92bf6e96273d24dd25807554de152e65fa727
3818bb3f013a73c

N=0xa05bb3783769b832a2d022646c48344948282cdcd42bff414ec90f23b7e3b1b81713766
4f4016319586395741996245f9c2c66dee453352dc329fe54228beaa559a610114dbe902c32
572e954660adbd06f8da8c770c33bb5ad15f506073ea0c50ff4e9906e16ee70d1311e0ad8189
6f4807282361f5b2116488de06966b571cdb15da536226378bc1fba8a3476c5809b5a274a01
17b5de3e52278d39fdfa62de29f338b0453ac3af61a30dcb2975949a3d0ec2d2b7f0d2c4d2e3
ef6ddefa8caad21bc16972dcecfcd5f9332373a759632f7f02c52dd424b83985eaa673ce67023
366e85899729fc1d1fede02fa9c53aa01328c9108a3c5145f47ef988688f3076d49821314210d
1f4db88fa836d41f3dc3960499eb46b28261aaa1515e0fb6d7481ae051b607683cbfdc18d6b6
92f93d6facf4002d6fa835aac4d61911b66859a81043763e1d0ef6e47f1a7a4c8d57993b0fb67
```

```
b5758ed3aca9540d39e150935cdd0c320d166da65612ae78322f96853885e6a44add306a89
9fab2f87cf2a1d

a = N+1-phi
f = x^2 - a*x + N
print(factor(f))
```

Results:

```
(x-
1725234152619972034714256060017298021802447336827305841926551733171440446
1805412050890170771347664013088369358363250522752072601460820803690310488
0675394240765781669186052702614848999356081231448357450693005935169350927
0597429187620138875718225578861813390897498641011726986749345160490062421
6381615228332290605504645975024029530764929502400538035021612093055725181
5536147337797291117242034871430920511816019822923729639314593907840925939
0252585051848041875663167)*(x-
2109355889745199958149438300501158694044426098698833969162385645905066414
0963100222872293597305255177751735799651200435654505848950353637094814800
2766954984754631961009692834502973691794751749585331730241120866779836454
0288806178300139693866176664932796506010586130267956850168651158391510613
1852411157772182928762035449606844199555188243420532483491035123230208568
4170864154270789165801519745352804523488963777629648703770859548188654191
1599438591088799785884067)
```

By comparing the results with (x-p)(x-q), we found that

p =
1725234152619972034714256060017298021802447336827305841926551733171440446
1805412050890170771347664013088369358363250522752072601460820803690310488
0675394240765781669186052702614848999356081231448357450693005935169350927
0597429187620138875718225578861813390897498641011726986749345160490062421
6381615228332290605504645975024029530764929502400538035021612093055725181
5536147337797291117242034871430920511816019822923729639314593907840925939
0252585051848041875663167

q =
2109355889745199958149438300501158694044426098698833969162385645905066414
0963100222872293597305255177751735799651200435654505848950353637094814800
2766954984754631961009692834502973691794751749585331730241120866779836454
0288806178300139693866176664932796506010586130267956850168651158391510613
1852411157772182928762035449606844199555188243420532483491035123230208568
4170864154270789165801519745352804523488963777629648703770859548188654191
1599438591088799785884067

Therefore, the factors of N after converting to hexadecimal are
0xb73cd57e66a1b78bcc5d2cf767ae94a617b2e1c70bd618c2e9fef928111d59259982d7e7a
a8d70b86cc53ce109cb7d50b166d56e770f917cb8fa6894e78849f2ce8ef4e463c448d14869e
37c0f523ea2056855283a859bf5ab24a3866acf1b5bf6ee2a31831cec861f75a9d91c37d04b6
e8d133413ab2c8f0f7f66c40e049a647f7157a66e77d58d36cadadc10419b228cdbfb4858642f
790d2cf87794cb5eff204d0c66ef77085a77d315932dd15ae9066bff9f3e72063c4ffbfcf3dd506
93f

and

0xe0090a82f4665efc3dee6cce9ee457a00ada319e3818283b86587313464c22b9b30db4da7562b2ead09f900058e9581798d7dbc84ee4fccb5972fe37a7999062c03837f97db2b60e03887f235033264a6f9d18b88b640026d99b17a99f921a99b5b87169a2ff5334e2da36672b3eeeb087451768dc403155ce52ea414226ea96afd455fc7e08d72fd1a672febccaaa2f1ff0135cf59b00b80c251692bbc0bcf92f0994f57c2fd28c6fa971a7df81161bbcc508efde052fbbc6222287ba6b19a3