Cyber Security Project

Task 2

2(a) Alice wants to use the "Hybrid" encryption method combining RSA and AES-128 in 8-bit CFB mode (CFB-8) to send an encrypted file to Bob (email address: bob@fit2093.edu). Given Bob's public key file bob.pub (this file is available on Moodle), Alice's AES-128 session key K=0x6cfba139ea2c55e5ecff60429ce20ade, CFB-8 IV value IV=0x6b66a2ce1972853f84d1874736369036, and the plaintext file msg qa.bin (this file is available on Moodle), compute the two components of the hybrid encryption ciphertext in two files: (1) the RSA ciphertext component in file c rsa.bin and (2) the AES ciphertext component in file c aes.bin, both files written in hexadecimal format. Show your code / working process.

Hybrid encryption involves encrypting the symmetric key K using RSA and sending it to the other person, then sending them the message encrypted with K with symmetric encryption. The steps involved are:

1. Select key K
2. Get the public key of the recipient
3. Encrypt K using the recipient's public key
4. Encrypt message using K and AES-128-CFB8 mode with specified K and IVs

Commands in blue below, with K in K.txt file:

First we import Bob's public key and encrypt K:

```
gpg --import bob.pub
gpg --fingerprint gpg
--sign-key
gpg --encrypt --output c_rsa.bin --recipient bob@fit2093.edu K.txt
```

Then we encrypt the msg_qa.bin with AES 128 CFB 8bit mode using K and the specified IV:

```
openssl enc -aes-128-cfb8 -e -in msg_qa.bin -out c_aes.bin -K
6cfba139ea2c55e5ecff60429ce20ade -iv 6b66a2ce1972853f84d1874736369036
```

To change the c_rsa.bin file from binary to hex:

```
xxd -plain c_rsa.bin > c_rsa.hex
xxd -plain c_aes.bin > c_rsa.hex
```

Result: c_rsa.bin:

c_aes.bin:

2(b) Bob received the two ciphertext component files c rsa qb.bin and c aes qb.bin encrypted by the hybrid encryption scheme in (a) from Alice. (NOTE: these files are available on Moodle and are different than the ones you have computed in part (a) of this task.) Given Bob's private key file bob.prv (this file is available on Moodle) and IV=0x5fe4bbaf52dfd660407a9a8e123901ea, show Bob's decryption process and the decrypted plaintext. Note: Bob's gpg passphrase is fit2093

Decrypting hybrid encryption steps are:

1. Decrypt the RSA cipher using Bob's private key
2. Convert the result to hex, as it was encrypted as binary
3. Using the resulting K to decrypt the message using AES-128-CFB8 along with specified IV.
4. Convert this result to hex.

Commands in blue below:

Import Bob's private key and decrypt K from c_rsa_qb with Bob's gpg passphrase: gpg --import bob.prv
gpg --output rsa_dec.txt --decrypt c_rsa_qb.bin
Enter passphrase:fit2093

We need to convert K from binary to hexadecimal:
xxd -p rsa_dec.txt
> 86a8c9f96dfe66bf6e7eaab5715be445

Decrypt c_aes_qb.bin using the above obtained K with AES 128 CFB 8bit mode and the specified IV:
openssl enc -aes-128-cfb8 -d -in c_aes_qb.bin -out msg.bin -K 86a8c9f96dfe66bf6e7eaab5715be445 -iv 5fe4bbaf52dfd660407a9a8e123901ea

We get the message as msg.bin.
So we convert msg.bin into hex:
xxd -p msg.bin

Result:

```
4f0cdf938bd6390fa676b44d9890d7220ba6264c940f06552a51f47c3a75
51475ae96a75d8f829982b0e83f3c570b983ebf17f859a3ab6041e40bebf
52c929ecda1bf40a0bd44a6241efd66705b83b1a9a86b182a9cccfef1f2a
7c7ebb89686c21425b0ff8c1303e4fe3962e5dae01f6266d8f38dd2c098f
1ca5e0e17370f66d757f091f3bc151387980dfecd6d9f99b1ec3579566e1
3c569ac6a8f2dbbb80de7b63c2975af1453c968b5a315caaed768f6738c9
29f7e2f9887b02270f385a01ba7e962adfaa4f0a6b160471bffd7adf2d70
b7763a6a450c69c20a11eae8ed5ef8e98651c21e29bbfb81abf6a2e2769f
2b8fe57a54644b6ce378b7b97098d66a
```

2(c) Due to network errors, instead of the files in part (b) of this task, Bob actually received the file c aes qc.bin, which contains an error in the 2nd block (this file is available on

Moodle). Compute how many blocks are corrupted by using a mathematical formula. Verify your result by decrypting c aes qc.bin.

Since cfb is in 8 bit mode a block refers to 2 hex characters, and since the 2nd block is corrupted, the 128 bit register will take 16 rounds, to reset, therefore there are 2 initial corrupted hex character plus the 16*2(since 2 hex characters are effected) =32 hex characters which is a total of 34 hex characters that are corrupted.

```
Openssl aes-128 -cfb8 -d -k 6cfba139ea2c55e5ecff60429ce20ade -iv
5fe4bbaf52dfd660407a9a8e123901ea -in c_aes_qc.bin -out verifying_blocks.bin
```

```
xxd -p verifying_blocks.bin
```

```
4f0d9f88c7ea79016e5f355ae8bc8ac04ca5264c940f06552a51f47c3a75
51475ae96a75d8f829982b0e83f3c570b983ebf17f859a3ab6041e40bebf
52c929ecda1bf40a0bd44a6241efd66705b83b1a9a86b182a9cccfef1f2a
7c7ebb89686c21425b0ff8c1303e4fe3962e5dae01f6266d8f38dd2c098f
1ca5e0e17370f66d757f091f3bc151387980dfecd6d9f99b1ec3579566e1
3c569ac6a8f2dbbb80de7b63c2975af1453c968b5a315caaed768f6738c9
29f7e2f9887b02270f385a01ba7e962adfaa4f0a6b160471bffd7adf2d70
b7763a6a450c69c20a11eae8ed5ef8e98651c21e29bbfb81abf6a2e2769f
2b8fe57a54644b6ce378b7b97098d66a
```

By comparing these file verifying_blocks.bin with the original c_aes_qc.bin, it can be seen that:
0d9f88c7ea79016e5f355ae8bc8ac04ca5 34 characters are corrupted so those 34 bits meaning 17 blocks meaning 2 to 18 blocks are corrupted.

2(d) The attacker Marvin intercepted two ciphertext files c aes1 qd.bin and c aes2 qd.bin sent by Alice. Marvin found out that due to a vulnerability in Alice's encryption software, Alice encrypted both files by using the same AES-128 key K and same IV, and he also managed to get the plaintext file msg1 qd.bin corresponding to ciphertext file c aes1 qd.bin by hacking Alice's computer, though he didn't find the second plaintext file msg2 qd.bin corresponding to the ciphertext file c aes2 qd.bin. Explain how Marvin can use his available information to find some information on the second plaintext file msg2 qd.bin corresonding to ciphertext c aes2 qd.bin, even though Marvin does not know Alice's encryption key K. Show the content of the plaintext blocks of msg2 qd.bin that Marvin can compute.

Reusing IVs creates a vulnerability where if 2 ciphertexts with the same IV are XOR'd then the result will reveal plaintext from both of the original plaintexts. In this case because IVs are shared Marvin can get both plaintexts by XORing their 2 ciphers together. The result of this is the plaintexts of both messages in one file. Using this result and his knowledge of the contents of msg1 he is able to get msg2 by XORing his msg1 file with the new file. However because CFB was used only the first blocks are vulnerable to this procedure as only that block will use this IV, the rest are based on ciphertext of the previous block. So we are only able to determine the first few digits of msg2, in this particular case 2 hexadecimal characters.

These are all mathematical steps that Marvin can use to get the plaintext file:

p1=msg1_qd.bin and p2 = msg2_qd.bin c of the two messages P1=c_aes1_qd.bin and P2=c_aes2_qd.bin. -It can be calculated that c1 = p1 xor K and c2 = p2 xor K, where K is the same key.

-If the two ciphertext blocks are xored against each other it produces c1 xor c2 = p1 xor p2. This allows for calculation of the first block of plaintext in the unknown message by solving for p1' due to knowing the value of the other three variables.

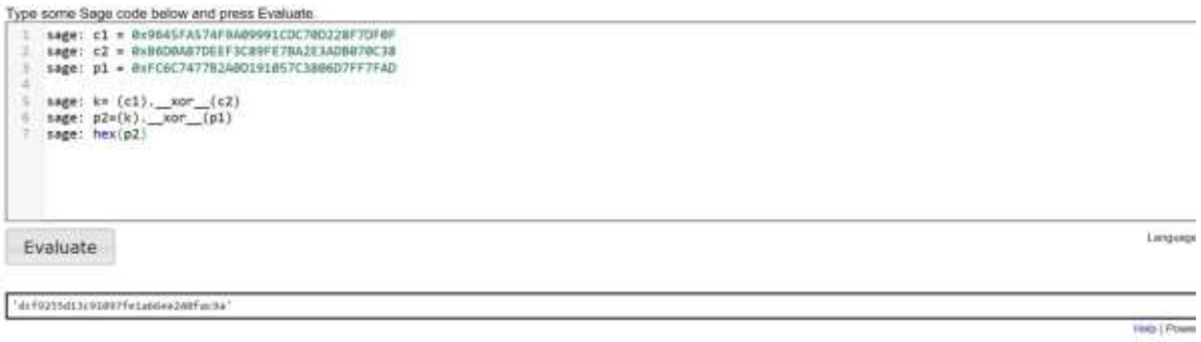
c1 = 96 45 FA 57 4F 9A 09 99 1C DC 70 D2 28 F7 DF 0F c2

= B6 D0 AB 7D EE F3 C8 9F E7 BA 2E 3A DB 07 0C 38 p1 =

FC 6C 74 77 B2 A0 D1 91 05 7C 38 06 D7 FF 7F AD

Therefore, p2 is equal to c1 xor c2 and xoring that result to p1 and we get p2: P2

= DC F9 25 5D 13 C9 10 97 FE 1A 66 EE 24 0F AC 9A


Type some Sage code below and press Evaluate

```
sage: c1 = 0x9645FA574F9A09991CDC70D228F7DF0F
sage: c2 = 0xB6D0AB7DEEF3C89FE7BA2E3ADB070C38
sage: p1 = 0xFC6C7477B2A0D191057C3806D7FF7FAD

sage: k= (c1).__xor__(c2)
sage: p2=(k).__xor__(p1)
sage: hex(p2)
```

```
'dcf9255d13c91097fe1a66ee240fac9a'
```

The only valid characters are DC because in CFB is 8 bit and the value of CFB remains same for the first block. Also, it provides a massive security vulnerability for data or any information being leaked. By using these above methods, the first block is fully decrypted and it happens because of using same IV vector.