پوریا غفوری ۴۰۱۱۰۶۲۷۷

سوال اول:

الف)

برای محاسبه ضریب انطباق در زبان فارسی فرمول زیر را داریم:

$$IC = \frac{\sum_{i=1}^{32} f_i(f_i - 1)}{N(N-1)}$$

برای محاسبه ضرایب انطباق بر اساس ترتیب حروف فارسی عمل کرده و اگر حرفی در عبارت ما نبود، آن را رد می‌کردیم.

برای عبارت اول داریم:

$$\frac{1\times0+2\times1+2\times1+1\times0+1\times0+2\times1+1\times0+1\times0+3\times2+1\times0+3\times2+1\times0+1\times0+1\times0+1\times0+1\times0+1\times0}{25\times24}$$

$$= \frac{18}{25\times24} = \frac{3}{100} = 0.03$$

برای عبارت دوم داریم:

$$\frac{3\times2+1\times0+1\times0+1\times0+2\times1+1\times0+1\times0+1\times0+1\times0+2\times1+1\times0+1\times0+1\times0+1\times0+1\times0+1\times0+1\times0+1\times0+2\times1}{25\times24}$$

$$= \frac{12}{25\times24} = 0.02$$

به طور کلی در متن‌های عادی، ضریب انطباق از عبارات تصادفی بیشتر است و دلیل آن این است که در عباراتی که معنادار هستند معمولا برخی از کاراکترها بیشتر تکرار می‌شوند و در نتیجه ضریب انطباق بالاتر می‌رود. در این مسئله با توجه به اینکه طول کلید ما نیز از عبارت ما بسیار کم‌تر است، می‌توان نتیجه گرفت که ممکن است کاراکترهای تکراری تولید کند و در نتیجه می‌توان حدس زد که عبارت اول که ضریب انطباق بیشتری دارد، عبارت رمزشده ما می‌باشد.

حال برای اینکه جایگاه عبارت بخشدوم را در عبارت تشخیص دهیم، توجه داریم که کاراکتر "ب" و "و" که به اندازه مضربی از کلید با یکدیگر فاصله دارند پس باید توسط یک حرف یکسان به دست آمده باشند. این سناریو برای عبارت "خ" و "م" نیز این قضیه برقرار است. پس بر اساس این الگوریتم می‌توان نتیجه گرفت که کاراکترهای معادل آن‌ها با یکدیگر ۲۸ واحد اختلاف دارند (برای ب و و) و برای خ و م نیز باید ۱۹ واحد اختلاف داشته باشند. بعد از بررسی عبارت متوجه می‌شویم که بخشدوم در جایگاه زیر قرار دارد:

چ	چ	ی	ع	ش	ع	ش	ع	ش	ر	ح	ب	ف	س	ژ	ر	ی	ق	غ	ب	ذ	ن	ش	ا	گ	ض
				ب	خ	ش	د	و	م																

می‌دانیم که اختلاف "ر" و "ش" اختلافی ۴- واحدی دارند که باقی‌مانده بر ۳۲ می‌شود ۲۸ و معادل چیزی است که قبلا بیان‌شده. همچنین برای "ح" و "ع" اختلاف برابر با ۱۳- که باقی‌مانده برابر با ۱۹ که معادل چیزی است که بیان شد.

حال می‌توان کلید را محاسبه کرد. برای کاراکتر اول کلید $x - 15$ باقی‌مانده بر ۳۲ برابر با 1 می‌شود که می‌توان دید کاراکتر اول برابر با س می‌شود. برای دومی $x - 20$ باقی‌مانده بر ۳۲ برابر با ۸ می‌شود پس کاراکتر دوم ز است. برای سومی $x - 15$ باقی‌مانده بر ۳۲ برابر ۱۵ می‌شود پس کاراکتر سوم الف است. برای چهارمی $x - 20$ باقی‌مانده بر ۳۲ برابر ۹ می‌شود پس کاراکتر چهارم ر است پس در کل کلید ما سزار می‌شود. حال عملیات کدگشایی را بر اساس این کلید بر روی همه کاراکترها انجام می‌دهیم (دقت کنید در محاسبه کلید منطقا باید کاراکترهای فارسی 0-index در نظر گرفته شوند):

برای محاسبه عبارت کدگشایی صرفا باید هرکاراکتر cipher منهای عبارت کلید معادلش شود و باقی‌مانده آن بر ۳۲ محاسبه شود.

cipher	ض	گ	ا	ش	ن	ذ	ب	غ	ق	ی	ر	ژ	س	ف	ب	ح	ر	ع	ش	ع	ش	ع	ی	چ	چ
Key	س	ر	ا	ز	س	ر	ا	ز	س	ر	ا	ز	س	ر	ا	ز	س	ر	ا	ز	س	ر	ا	ز	س
plain	ت	س	ا	ت	س	ی	ب	د	د	ع	ر	ب	ا	ر	ب	م	و	د	ش	خ	ب	د	ی	ل	ک

ب)

عملیات کدگشایی برای عملیات سزار به این صورت است که کاراکترهای عبارت باید هر کدام منهای کلید شوند و باقی‌مانده بر ۳۲ محاسبه شود و سپس کاراکتر جدید جایگزین آن شود.

مثلا برای کاراکتر اول ف داریم $22 - 20 \ \% \ 32 = 2$ پس ف تبدیل به پ می‌شود. همین روند را ادامه می‌دهیم و به عبارت "پاسخ سوال اول تکمیل شد" می‌رسیم.

سوال دوم:

الف)

$$S_8(x_1) = 13, S_8(x_2) = 01$$

$$S_8(x_1 \oplus x_2) = S_8(x_2) = 01$$

$$S_8(x_1) \oplus S_8(x_2) = 12 \neq 01$$

پس برای این مورد شرط غیرخطی بودن برقرار است.

ب)

$$S_8(x_1) = 11, S_8(x_2) = 07$$

$$S_8(x_1 \oplus x_2) = S_8(011111) = 02$$

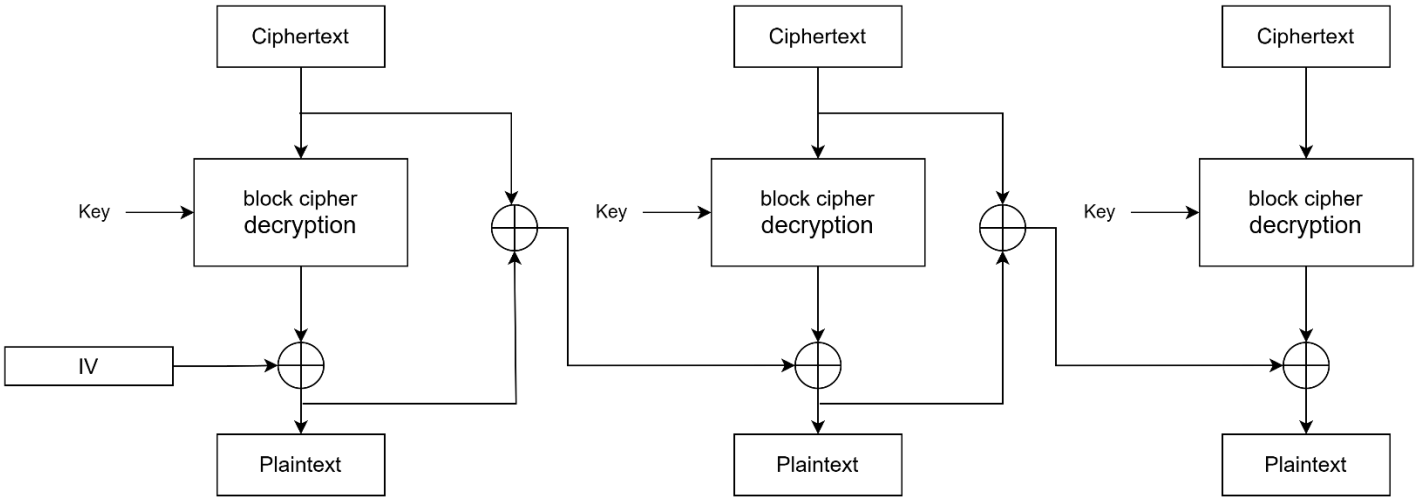
$$S_8(x_1) \oplus S_8(x_2) = 12 \neq 02$$

پس برای این مورد هم شرط غیرخطی بودن برقرار است.

سوال سوم:

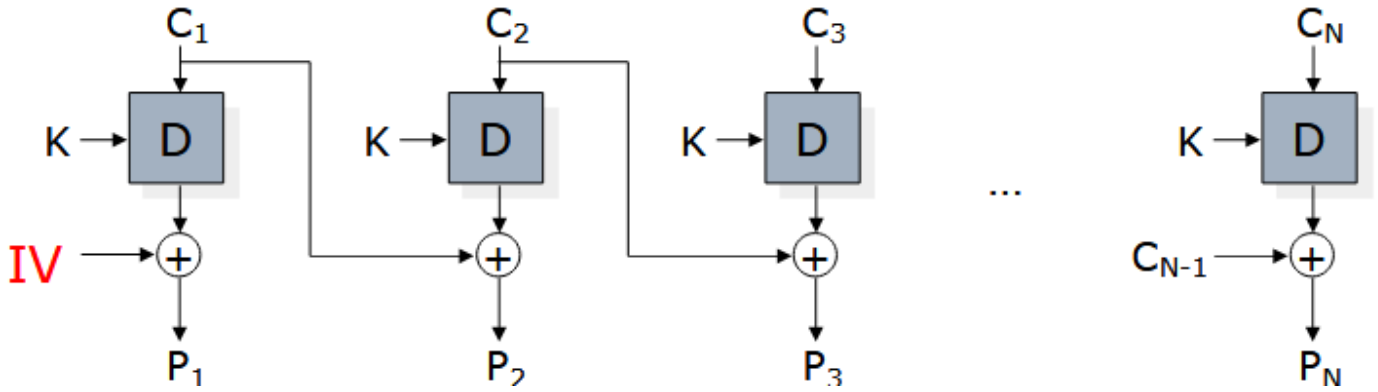
الف)

ابتدا فرایند کدگشایی PCBC را در شکل زیر نمایش می‌دهیم:



همان‌طور که در تصویر بالا مشخص است، در صورتی که یکی از متن‌های رمزشده به مشکل بخورد، چون در فرایند رمزگشایی بلوک‌های بعدی تاثیر دارد (همچنین plaintext خودش نیز اشتباه می‌شود) پس باعث می‌شود که مقدار plaintext در بلوک بعدی به مشکل بخورد و با به مشکل خوردن plaintext در بلوک بعدی، این سلسله مراتب تکرار شده و تا انتها همه plaintextها اشتباه می‌شوند.

فرایند کدگشایی CBC:



در این فرایند اگر یک ciphertext به مشکل بخورد، صرفا plaintext خودش و plaintext بلوک بعدی به مشکل می‌خورد و چون plaintextها در فرایند کدگشایی تاثیری ندارند (برعکس حالت PCBC که plaintextها در فرایند کدگشایی بلوک‌های بعدی تاثیر دارند) پس فقط plaintext همان بلوک و بلوک بعدی به مشکل می‌خورد و از آن به بعد بقیه plaintextها درست خواهند بود.

از مزایای PCBC می‌توان به چند مورد اشاره کرد:

مورد اول زمانی است که حمله افزایش طول صورت می‌گیرد. در حالت CBC، حمله افزایش طول به راحتی قابل انجام است چون در عمل ما داریم $D_N \oplus P_1$ را به جای P_1 جا می‌زنیم و در نتیجه محتویات بعد xor تغییر نمی‌کند و فرایند کدگذاری به درستی انجام می‌شود. اما در حالت PCBC با توجه به اینکه هم P1 قبل از Xor و هم بعد از Xor در فرایند کدگذاری تاثیر دارد و قطعا یکی از این موارد در افزایش طول تغییر کرده و در نتیجه پیام تا انتها خراب می‌شود و گیرنده در فرایند احراز صحت متوجه خواهد شد که پیام دستکاری شده است.

مورد بعدی البته زمانی است که ممکن است خطایی در انتقال داده به وجود آمده باشد که در این صورت در حالت CBC باز متوجه نخواهیم شد که متن به مشکل خورده است (فقط در حالتی که در دو بلوک آخر خطا باشد متوجه می‌شویم) اما در PCBC به دلیل اینکه خطا منتقل می‌شود، در هنگام احراز صحت پیام، گیرنده متوجه وجود خطا می‌شود.

ب)

برای این که رمزگشایی ادامه پیام (بعد از این دو قالب) به درستی انجام شود، نیاز است که مقداری که بعد از xor نهایی این دو بلوک به بلوک بعدی می‌رود، تغییر نکند. این مقدار را برابر با B می‌گیریم. (در اصل B برای محاسبه plaintext در بلوک سوم استفاده می‌شود)

در حالت اول فرض کنیم بلوک اول C_i و P_i است و مقداری که برای xor کردن (برای محاسبه نهایی plaintext) در این بلوک استفاده می‌شود برابر با A است و بلوک بعدی نیز شامل C_{i+1} و P_{i+1} است. داریم:

$$P_i = D(C_i, k) \oplus A$$

$$P_{i+1} = C_i \oplus P_i \oplus D(C_{i+1}, k)$$

$$B = P_{i+1} \oplus C_{i+1}$$

$$B = C_i \oplus D(C_i, k) \oplus A \oplus D(C_{i+1}, k) \oplus C_{i+1}$$

در حالت دوم که جای این دو جابه‌جا می‌شود (بلوک اول P_{i+1} و C_{i+1} و بلوک دوم P_i و C_i):

$$P_{i+1} = D(C_{i+1}, k) \oplus A$$

$$P_i = C_{i+1} \oplus P_{i+1} \oplus D(C_i, k)$$

$$B = P_i \oplus C_i$$

$$B = C_{i+1} \oplus D(C_{i+1}, k) \oplus A \oplus D(C_i, k) \oplus C_i$$

و همان‌طور که می‌دانیم *xor* خاصیت جابه‌جایی دارد و هر دو B که در این دو حالت محاسبه کردیم کاملا با یکدیگر برابر هستند پس در نتیجه سیگنال B که برای کدگشایی به بلوک بعدی می‌رود تغییری نکرده و در نتیجه plaintext بلوک بعدی تغییری نخواهد کرد.

سوال چهارم:

الف)

با توجه به اینکه اگر کارمند p_a را داشته باشد صرفا می‌تواند p سطرهای بالاتر خود را به دست بیاورد (با توجه به فرمول که هر p از p قبل خود محاسبه شده است) پس شخص می‌تواند از p_a تا p_b را با داشتن p_a محاسبه کند.

اگر کارمند k_b را داشته باشد، با توجه به اینکه p_a تا p_b را دارد پس می‌تواند با محاسبه $k_b \oplus p_b$ می‌تواند q_b را محاسبه و در نتیجه با توجه به فرمول محاسبه q_i فرد می‌تواند از q_b تا q_a را محاسبه کند و در نتیجه k_a تا k_b را خواهد داشت. با توجه به اینکه کارمند فقط می‌تواند pهای برای بازه بزرگتر از b را داشته باشد و نمی‌تواند pهای کوچکتر از a را داشته باشد و برای q هم فقط می‌تواند qهای کوچکتر از a را داشته باشد و نمی‌تواند qهای بزرگتر از b را داشته باشد.

در نتیجه برای بازه‌های بالاتر از b، کارمند q را ندارد و برای بازه‌های پایین‌تر از a، کارمند p را ندارد و در نتیجه نمی‌تواند k را محاسبه کند.

ب)

برای توضیح راحت‌تر بر روی یک مثال این مشکل را نشان می‌دهیم.

فرض کنید که کارمند A بازه ۱ تا ۳ را در اختیار دارد و کارمند B بازه ۶ تا ۹ را در اختیار دارد. (فرض کنید ۱ و ۹ کف و ته بازه ما هستند.

در نتیجه A از p_1 تا p_9 و q_3 تا q_1 را در اختیار دارد. B از p_6 تا p_9 و q_9 تا q_1 را در اختیار دارد.

این دو با کمک یکدیگر هم به p_4 تا p_5 (که از A می‌آید) و هم به q_5 تا q_4 (که از B می‌آید) دسترسی دارند و در نتیجه هر دو به k_4 و k_5 دسترسی دارند با اینکه هیچکدام از آن‌ها نباید آن‌ها را داشته باشد.

ج)

تنها حالت این است که k_a تا k_b را در اختیار کارمند بگذاریم. حال آن را توضیح می‌دهیم.

با توجه به اینکه کارمند باید مقدار k را محاسبه کند تا بتواند به یک سطر دسترسی داشته باشد پس باید حتما p و q مربوط به آن سطر را در اختیار داشته باشد. اگر کارمند p_i و q_i در اختیار داشته باشد باعث می‌شود که همه pهای بزرگتر از i و همه qهای کوچکتر از i را نیز در اختیار داشته باشد و در نتیجه سناریو بخش ب همچنان ممکن است اتفاق بیفتد. همچنین نمی‌توانیم یک k و یک p یا q در اختیار کارمند بگذاریم چون می‌توان از این طریق آن یکی p یا q (مشابه الف) را نیز محاسبه کرد و باز هم سناریو مشکل دارد. حالت دیگر این است که فقط یکی از p یا q را به کارمند بدهیم که در این صورت نیز خود کارمند نمی‌تواند به سطرهای خود دسترسی داشته باشد چون نمی‌تواند k را محاسبه کند. پس تنها حالتی که باقی می‌ماند این است که k را در اختیار کارمند بگذاریم.

سوال پنجم:

این سوال را با کمک کتابخانه wiener حل کردیم که در Q5/Q5.py موجود است. در نهایت نیز پس از اجرای آن به خروجی زیر رسیدیم:

```
@pourta > python3 Q5.py
d = 91536660463053297224401282133764817089761055337192521175666433938605479662717281610970877602326991045985211486855657913655204566051982582119782506253496 91
flag = Network_Security{B0n3h_Durf33_>_Wi3n3r}
```

طبق مقاله می‌توان متوجه شد اگر p و q آن‌گاه باید شرط زیر برقرار باشد:

$$if\ p > q\ then\ p < 2q$$

$$if\ q < p\ then\ q < 2p$$

به این معنا که این دو عدد تقریبا نزدیک یکدیگر هستند و شرط دیگر $3d < \sqrt[4]{N}$ است. در حالتی که این دو برقرار باشد حمله وینر موثر است. البته دقت کنید این اگر این شروط برقرار هم نباشد ممکن است این حمله موثر باشد (شروط کافی هستند نه لازم) و اگر دقت کنید در کد سوال شرط حتما بررسی می‌شود که شرط $3d < \sqrt[4]{N}$ نقض شود. در فرایند اثبات برای اینکه بتوانیم به عبارات زیر برسیم از $3d < \sqrt[4]{N}$ استفاده کردیم:

$$k\phi(N) = ed - 1 < ed, e < \phi(N) \Rightarrow k < d < \frac{1}{3}\sqrt[4]{N}$$

اما در اینجا خود شرط $k < \frac{1}{3}\sqrt[4]{N}$ می‌تواند برقرار باشد (چون اگر e به مقدار قابل توجی از $\phi(N)$ کمتر باشد، آن‌گاه این امکان وجود دارد که k نیز از $\frac{1}{3}\sqrt[4]{N}$ کوچکتر باشد).

حال پس از بررسی کد متوجه می‌شویم که d حداکثر عدد ۵۱۲ بیتی است و چون n و در نتیجه $\phi(N)$ حداکثر ۲۰۴۸ بیتی هستند پس می‌توان نتیجه گرفت که تقریبا همیشه d بسیار عدد کوچکتری نسبت به n و $\phi(N)$ خواهد بود و پس e همچنان از $\phi(N)$ بسیار کوچکتر خواهد بود پس همچنان می‌تواند شرط بالا برقرار باشد و به احتمال بالایی قضیه وینر در این مسئله برقرار خواهد بود.

برای کد این بخش هم از کتابخانه *owiener* استفاده کردیم تا d را محاسبه کند و سپس بتوانیم عبارت را کدگشایی کنیم.

```python
import owiener
from Crypto.Util.number import long_to_bytes

e = 0
n = 0
cipher_text = 0
with open("Q5.txt", "r") as file:
    for line in file:
        if "=" in line:
            name, value = line.strip().split(" = ")
            if name == "e":
                e = int(value.strip(), 16)
            elif name == "n":
                n = int(value.strip(), 16)
            elif name == "cipher_text":
                cipher_text = int(value.strip(), 16)

d = owiener.attack(e, n)

print(f"d = {d}")
print(f"flag = {long_to_bytes(pow(cipher_text, d, n)).decode('utf-8')}")
```

سوال ششم:

الف)

$$n = pq = 3 \times 7 = 21$$

$$\phi(21) = 2 \times 6 = 12$$

$$e = 5$$

$$d \times 5 = 1 \; [mod \; 12] \Rightarrow d = 5$$

البته d=5 مورد جالبی نیست چون در این صورت هم کلید عمومی هم کلید خصوصی با یکدیگر برابر می‌شوند پس سراغ d بعدی می‌رویم.

$$d \times 5 = 1 \; [mod \; 12] \Rightarrow d = 17$$

پس در کل کلید عمومی $\{5,21\}$ و کلید خصوصی $\{17,21\}$ است.

حال عملیات رمزنگاری را انجام می‌دهیم:

$$C = 10^5 \; mod \; 21 = 19$$

حال عملیات رمزگشایی را انجام می‌دهیم:

$$M = 19^{17} mod \; 21 = 10$$

حال دیدیم که عملیات‌ها به درستی انجام شد.

ب)

$$n = pq = 7 \times 17 = 119$$

$$\phi(119) = 6 \times 16 = 96$$

$$e = 11$$

$$d \times 11 = 1 \; [mod \; 96] \Rightarrow d = 35$$

پس در کل کلید عمومی $\{11,119\}$ و کلید خصوصی $\{35,119\}$ است.

حال عملیات رمزنگاری را انجام می‌دهیم:

$$C = 11^{11} \; mod \; 119 = 114$$

حال عملیات رمزگشایی را انجام می‌دهیم:

$$M = 114^{35} \; mod \; 119 = 11$$

حال دیدیم که عملیات‌ها به درستی انجام شد.

ج)

$$n = pq = 17 \times 23 = 391$$

$$\phi(391) = 16 \times 22 = 352$$

$$e = 9$$

$$d \times 9 = 1\ [mod\ 352] \Rightarrow d = 313$$

پس در کل کلید عمومی $\{9, 391\}$ و کلید خصوصی $\{313, 391\}$ است.

حال عملیات رمزنگاری را انجام می‌دهیم:

$$C = 7^9\ mod\ 391 = 61$$

حال عملیات رمزگشایی را انجام می‌دهیم:

$$M = 61^{313}\ mod\ 391 = 7$$

حال دیدیم عملیات‌ها به درستی انجام شد.

سوال هفتم:

الف)

با مشاهده‌ی ابتدای فایل متوجه می‌شویم که نحوه رمزگذاری آن به صورت DES-EDE3 و با مود CBC است و IV آن نیز برابر با عبارت F5CFD... است. این فایل بر این اساس کدگذاری شده است و صرفا اگر کاربر کلید این رمز را داشته باشد می‌تواند فرایند کدگشایی را انجام دهد تا به محتویات فایل دسترسی داشته باشد.

```
pouria@pouria:/mnt/u/Data and Network Security/Exercises/3/Q7$ head key.pem
-----BEGIN PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,F5CFD36048012B10

eiEMsx0D7t9W/7TnH21tw1ba84Is6A6hwAdBpFvOnXsOS4BJsb2Gzhu95wgNXQMy
EgSC6Tj3M0jRMHEXOAMNqcun0aBlLgAVoytxxZxHpe6vzL8Q6Oh3ylZWsHqaQFzS
EWdzP1RpruOIkVq6Re51dNh/LlFaGk+YwJPKJVN0e/B69qdWxw8HcpxoHU9hbL0u
/GZ+rA1mdjnEtZLtCLJiK6rupPCrj2mMtFNaOdS8VGskE5TCJmARmgnv9ORGEqFp
W9WOFoAN7brFIx9IOX233lCpTnxyZwcvQn9X70Zxs1jSWeXEox3rw6k5FqmMg6V5
L6Qg8GfcGp9TRSxvABNHDpi3Fwlf9OVDo9C6RbQn64JDeFQjiKSUJ0i74Kd29LsR
pouria@pouria:/mnt/u/Data and Network Security/Exercises/3/Q7$
```

ب)

برای محاسبه این مقدار، محتویات فایل key.pem را مشاهده کرده و p و q را از آن استخراج کرده و با استفاده از کد پایتون موجود در Q7/phi_n.py آن را محاسبه می‌کنیم:

```
d = 21805295917455307723766

p = 31474892141467783459896
q = 24583341282572439616761

phi_n = (p - 1) * (q - 1)

print(phi_n)
```

```
773758015345859820397677045666033786172146042019761744911254659055304915038566087240887584399
832642735119286182328560684959021320321077435385699433913191536694397502349490257358284428561
269128062832305986482073448592573912775509158910809548196052281780913074889693910114986942108
544940039100172945046432516109997785308157099031498640186241129883383811917602606344164999796
906572943397983368224558762800321307022320510654316706975096456269055233768064770365225038905
8909450014980
```

ج)

با استفاده از دستور زیر، این کار را انجام می‌دهیم:

```
@pouria  openssl rsautl -decrypt -inkey key.pem -in encrypted_message.enc -out file.dec
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
Enter pass phrase for key.pem:

Mon 19 May - 22:35  /run/media/pouria/University/Data and Network Security/Exercises/3/Q7
@pouria  cat file.dec
A secure network is like a chain - only as strong as its weakest link.%
```

و همان‌طور که مشاهده می‌کنید محتویات به درستی کدگشایی شده است.

د)

ابتدا با دستور زیر می‌توانیم کلید عمومی را استخراج کنیم:

```
@pouria openssl rsa -in key.pem -pubout -out public_key.pem
Enter pass phrase for key.pem:
writing RSA key

Mon 19 May - 22:39 /run/media/pouria/University/Data and Networ
@pouria cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvanCHlcVZcRrkAJW3l84
lB1xAuYEyT0PFOgRvQz2L07K+kEV/AD9pJjSEdfb+9105ZIiEGCavtpYKgJEggsO
tWRUW1FvdxKhqywPJqsmquabi3xfnd3BCJAJCOkWeV7HnD3aJCxrDm9/QfVPefX7
28g1p7ipnUJx+R7j2lJLhLQXwbHqDLmSNCdTumZ2CSjIxM4JeuEnS9L/T1UH0vB2
5/VPeaCnd4ESSnvsa3wmJEMCseiDHE6Bkxr/WjJWHPsG9+fXsg5USMbs5uH1cK6g
synRVN8Eg5OrhpUTKDAzUmerPgoa3sZzBski5bJrQ+PHbP9+yr9SbMyZ0ZOCoW2/
/Y17m3iDSrjMXublpP9wvgYA4OU9xwhGLSCOb3DQilShoxfd11kKEHG4Mm233q/z
VxucL8wSg3C6BD3i5TxnF+9wVFqFYKZ0HWjZVyTTGmcg6XkWt3AMaynrXce3Aw3H
u3J2/hzHfx80ZjPiARHrEcSkimfZorE1MZpdUE4aybChZNUYWX0qC2g8eGhmMf31
+U1htKOYRv/78YeWh3rWXqMbH3ihXtY+qIwd1OTIBJQC3jEbGM33WZLzx/4iblSs
i4206lRGsmWujv8/PKX5fKtwKthKOdCjq37qtVQ6Gwr91FRm7HlLg95CFjzgJPZG
ftb3cWeQIFG635c9WTgycAECAwEAAQ==
-----END PUBLIC KEY-----
```

با دستور زیر نیز می‌توانیم فایل کدگشایی شده را با استفاده از کلید عمومی دوباره رمز کنیم:

```
@pouria openssl rsautl -encrypt -inkey key.pem -pubin -in file.dec -out file.enc
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
Enter pass phrase for key.pem:
Enter pass phrase for key.pem:

Mon 19 May - 22:38 /run/media/pouria/University/Data and Network Security/Exercises/
@pouria cat file.enc
kW◆◆◆◆BIIA◆◆ZJ?2◆.◆T◆◆◆◆?TX◆ub◆^|◆◆G3◆◆cC◆uE&\U$F&◆o`◆◆◆m◆◆oz|f◆◆◆햇◆◆◆sfaZ◆I◆>8@
                                                                    0[◆F◆◆
◆◆ ◆u◆
/T@◆WQ◆◆Xи◆;◆◆◆◆R)◆◆3жu ◆-◆<◆◆◆iM=◆◆◆'S◆
                                    ◆◆◆◆◆◆6!◆◆$v◆[◆B&◆◆4^◆◆◆◆Q◆!G◆QH`h◆cے/Y_◆◆
                                                                    ◆◆M◆◆
(◆◆◆◆xOS*F◆◆Q`◆◆◆◆◆%◆◆◆lM◆n◆'◆◆Z◆◆◆◆@◆◆p◆◆◆◆◆gJ2◆pw%
Mon 19 May - 22:38 /run/media/pouria/University/Data and Network Security/Exercises/
@pouria cmp file.enc encrypted_message.enc
file.enc encrypted_message.enc differ: byte 1, line 1
```

همان‌طور که مشاهده می‌کنید بعد از اجرای دستور cmp، نشان می‌دهد که این دو فایل در همان بایت اول یکسان نیستند و در نتیجه محتویات فایل‌ها یکسان نیست.

دلیل این امر این است که رمزگذاری RSA از padding استفاده می‌کند که مقادیر تصادفی داخل داده قرار می‌دهد تا هر دفعه خروجی الگوریتم با داده یکسان در کدگذاری یکسان نخواهد شد.

ه)

در ابتدا تلاش کردم تا یک ویدیو که نسبتا حجم بزرگی داشت را با استفاده از کلید عمومی رمزگذاری کنم:

```
@pourla ❯ openssl rsautl -encrypt -inkey public_key.pem -pubin -in ce441-hw2-4.mkv -out file.enc
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
RSA operation error
803BA74CAC7B0000:error:0200006E:rsa routines:ossl_rsa_padding_add_PKCS1_type_2_ex:data too large for key size:crypto/rsa/rsa_pk1.c:133:
```

اما همان‌طور که مشاهده می‌شود با خطای بزرگ بودن فایل مواجه شدیم. دلیل این امر این است که برای رمزگذاری به شیوه RSA ما باید محتویات خود را در بازه کوچکتر از n نگه داریم تا بتوان فرایند رمزگذاری را انجام داد. روش پیشنهادی این است که از یک روش رمزنگاری نامتقارن استفاده کنیم و با استفاده از رمزنگاری متقارن، کلید خود را کدگذاری کرده و با فرد مورد نظر به اشتراک بگذاریم چون کلیدها معمولا طول کمتر از n دارند و می‌توان با روش متقارن بدون مشکل آن‌ها را رمز کرد اما رمزنگاری های نامتقارن محدودیت حجم نداشته و حتی فایل‌های بزرگ را می‌توانیم به اشتراک بگذاریم.

سوال هشتم:

کد این سوال در Q8/Q8.py موجود است. هر دو کد داخل این فایل هستند و شما می‌توانید با اجرا کردن این برنامه و وارد کردن شماره بخش سوال (۱ یا ۲) کد مربوط به آن بخش را اجرا کرده و خروجی آن را مشاهده کنید.

الف)

با توجه به تابع رمزگذاری، می‌توانیم تابع رمزگشایی معادل آن را بنویسیم:

```python
def dec_mac(k, c, t):
    # decrypt
    m = AES.new(mode=AES.MODE_OFB, key=k, iv=k).decrypt(c)
    # MAC
    t_dec = AES.new(mode=AES.MODE_CBC, key=k, iv=c[:BLOCK_SIZE]).encrypt(m)[-BLOCK_SIZE:]

    pad_size = m[-1]
    pad_size = int(pad_size) if int(pad_size) != 0 else BLOCK_SIZE
    m = m[0:len(m) - pad_size]
    return m.decode('utf-8`'), t == t_dec
```

در این کد ابتدا c کدگشایی شده و m به دست می‌آید و سپس بر اساس m، مقدار t را محاسبه کرده (t_dec) تا عملیات احراز صحت پیام انجام شود. (پارامتر دوم return شرط برابری t_dec و t دریافتی را بررسی کرده تا متوجه شود پیام به درستی

دریافت شده یا خیر). در ادامه می‌دانیم که بایت آخر اندازه pad را ذخیره کرده پس آن را استخراج کرده و از انتهای پیام حذف می‌کنیم. (البته شرط بررسی ۰ بودن آن در این تابع بی‌فایده است چون با توجه به تابع enc_mac، هیچ‌گاه مقدار pad صفر نخواهد بود و همیشه مقداری میان ۰ تا ۱۵ خواهد داشت)

در ادامه این کد را به ازای دو ورودی سوال و کلید آن بررسی می‌کنیم:

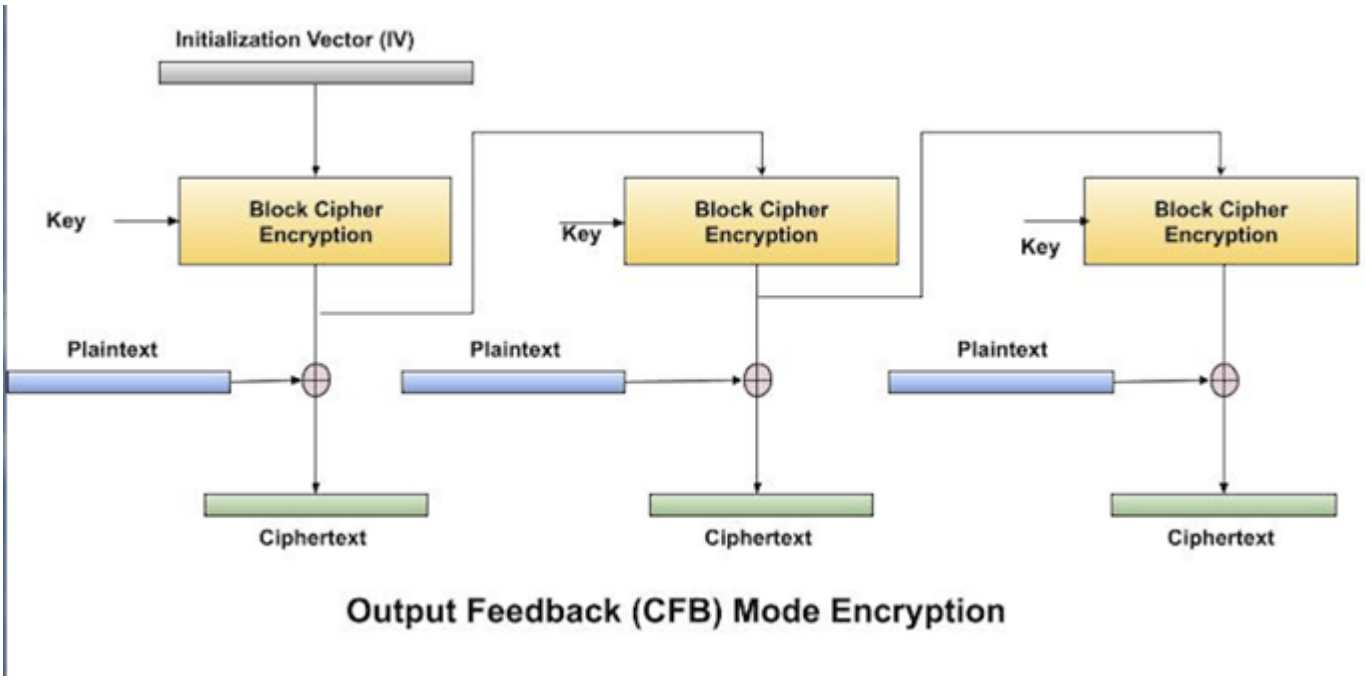
```
@pouria python3 Q8.py


Choose part: 1
For (c, t) = ('d8b8239628a3f44c81e50cbd57aaac62586cdf1376c25fa8c23e8becf6be4688', 'abb859c60dd1450bd789a40bc3638f4e')
result:
('enemy knows the system', True)

For (c, t) = ('f8a0238928a3fc4b9efa1aef03aaa62e4f668c0633dc21cdba4dafe3f9b14987', 'b893a8d5032f5c004f11543626fc942e')
result:
('Every cipher holds a story.', False)



Choose part: 2
New exploited cipher text = fa6f5a64fe6075eff617ee2cf2c715ef
Verification = ('everything is o', True)
```

همان‌طور که مشاهده می‌کنید برای بخش اول سوال هر دو رشته به درستی کدگشایی شده‌اند و همان‌طور که مشاهده می‌کنید احراز صحت برای عبارت اول تایید شده اما صحت عبارت دوم تایید نشده است.

ب)

برای این بخش ابتدا باید بررسی کنیم که فرایند محاسبه t برای عبارت delete all keys به چه صورت است. فرض می‌کنیم که این عبارت بعد از padding (چون طول آن ۱۵ است پس یک واحد پدینگ دارد و یک بایت با مقدار ۱ به انتهای آن اضافه می‌شود) برابر با P_1 است و مقدار کدگذاری شده در سوال برابر با C_2 می‌باشد.

بر اساس حالت OFB داریم:

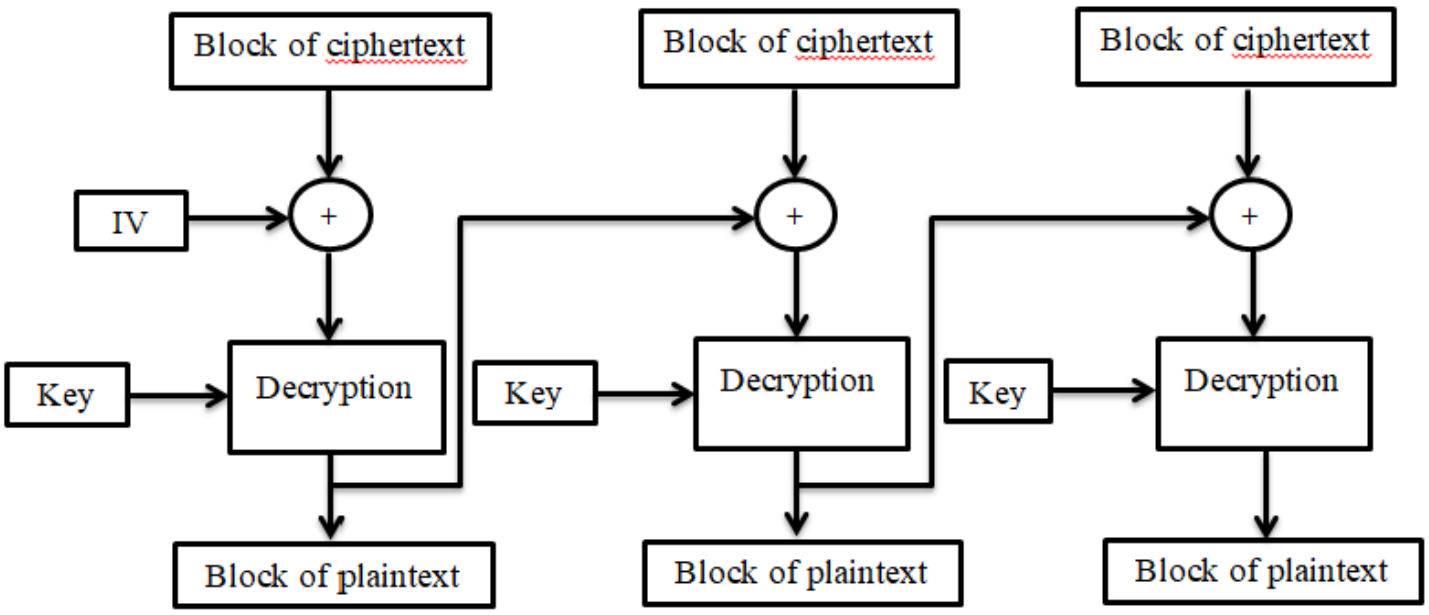


Output Feedback (CFB) Mode Encryption

حال با توجه به این مدل و اینکه IV=k برای محاسبه C_1 خواهیم داشت:

$$C_1 = E(k,k) \oplus P_1$$

حال برای محاسبه t با توجه به اینکه حالت *CBC* را داریم:



با توجه به اینکه طول بلوک ما برابر با ۱۶ است و طول P_1 هم برابر با ۱۶ شده پس برای محاسبه t نیز از همان بلوک استفاده می‌کنیم و مطابق با کد *enc* داریم:

$$IV = E(k,k) \oplus P_1$$

$$t = E(E(k,k) \oplus P_1 \oplus P_1, k) = E(E(k,k))$$

پس همان‌طور که دیدیم در این حالت، t کاملا مستقل از پیام آشکار و پیام رمزشده است و صرفا وابسته به الگوریتم و کلید است و ربطی به هیچ‌چیز دیگری ندارد. پس اگر ما به جای c هرچیزی بگذاریم که طول آن به اندازه طول بلوک (۱۶) باشد مشکلی پیش نخواهد آمد. پس به همین دلیل می‌توانیم عبارت ***everything is ok*** را با استفاده از کلید رمزکرده و به جای c قرار دهیم و در ادامه زمانی که گیرنده می‌خواهد t را محاسبه کند، با توجه به اینکه مستقل از همه چیز است (به غیر از الگوریتم و کلید) پس t را مانند همان t دریافتی می‌بیند و احراز صحت تایید می‌شود. دلیل این امر استفاده از دو مود ***CBC*** و ***OFB*** به طور همزمان است که باعث شده وابستگی t نسبت به داده از بین برود.

البته دقت کنید که با توجه به ویژگی ***padding***، ما نمی‌توانیم تمام ۱۶ بایت را به عنوان داده بدهیم چون آن‌گاه الگوریتم کدگشایی بایت آخر را به عنوان اندازه ***padding*** در نظر می‌گیرد و آن‌گاه بر اساس آن کدگشایی را انجام می‌دهد و منطقا برنامه به مشکل می‌خورد. به همین دلیل این امکان وجود ندارد که عبارت ***everything is ok*** کاملا منتقل شود و ما مجبوریم یک کاراکتر از آن حذف کنیم و سپس خودمان ***padding*** به آن اضافه کنیم و یک بایت یک را در انتهای آن قرار دهیم تا طول آن برابر با ۱۶ شود و الگوریتم هم به مشکل نخورد.

حال در ادامه فرض می‌کنیم که عبارت ***delete all keys*** به علاوه ***padding*** برابر با P_1 و عبارت ***everything is o*** به علاوه ***padding*** برابر با P_2 در نظر می‌گیریم. حال برای محاسبه C_2 که همان عبارت کدگذاری‌شده P_2 است داریم (C_1 عبارت کدگذاری‌شده P_1 است):

$$C_1 = E(k, k) \oplus P_1$$

$$C_2 = E(k, k) \oplus P_2$$

اما در اینجا ما k را نداریم تا C_2 را محاسبه کنیم پس بر اساس دو عبارت بالا می‌توان نتیجه گرفت:

$$C_1 \oplus P_1 = E(k, k)$$

$$C_2 = C_1 \oplus P_1 \oplus P_2$$

و بر این اساس C_2 ساخته می‌شود و می‌توان آن را جای عبارت کدگذاری‌شده اصلی فرستاد.

کد توضیحات بالا عینا در تصویر زیر مشخص است و کامل مطابق با توضیحات بالا زده شده و به نظر نیاز به توضیح ندارد.

```python
def exploit(c, t):
    valid_data = "delete all keys".encode('utf-8')


    invalid_data = "everything is o".encode('utf-8')
    r = BLOCK_SIZE - len(invalid_data) % BLOCK_SIZE
    pad_size = r if r != 0 else BLOCK_SIZE
    invalid_data += pad_size.to_bytes(1, 'big') * pad_size

    one = 1
    tmp = bytes(a ^ b for a, b in zip(valid_data + one.to_bytes(1, 'big'), c))
    new_cipher = bytes(a ^ b for a, b in zip(tmp, invalid_data))

    return new_cipher.hex()
```

برای مطمئن شدن از کد خود تابع زیر را تعریف کردیم که یک عبارت را با یک کلید فرضی کد کرده و سپس یک عبارت دیگر را جایگزین آن می‌کنیم تا مشاهده کنیم ***exploit*** ما به درستی اجرا می‌شود:

```python
def verification(k):
    valid_data = "delete all keys".encode('utf-8')

    invalid_data = "everything is o".encode('utf-8')
    r = BLOCK_SIZE - len(invalid_data) % BLOCK_SIZE
    pad_size = r if r != 0 else BLOCK_SIZE
    invalid_data += pad_size.to_bytes(1, 'big') * pad_size

    c_valid, t_valid = enc_mac(k, valid_data)
    one = 1
    tmp = bytes(a ^ b for a, b in zip(valid_data + one.to_bytes(1, 'big'), c_valid))
    new_cipher = bytes(a ^ b for a, b in zip(tmp, invalid_data))

    return dec_mac(k, new_cipher, t_valid)
```

در ادامه برنامه خود را اجرا می‌کنیم:

```
@pouria python3 Q8.py


Choose part: 1
For (c, t) = ('d8b8239628a3f44c81e50cbd57aaac62586cdf1376c25fa8c23e8becf6be4688', 'abb859c60dd1450bd789a40bc3638f4e')
result:
('enemy knows the system', True)

For (c, t) = ('f8a0238928a3fc4b9efa1aef03aaa62e4f668c0633dc21cdba4dafe3f9b14987', 'b893a8d5032f5c004f11543626fc942e')
result:
('Every cipher holds a story.', False)



Choose part: 2
New exploited cipher text = fa6f5a64fe6075eff617ee2cf2c715ef
Verification = ('everything is o', True)
```

همان‌طور که برای بخش دوم مشخص است، در ابتدا عبارت کدگذاری‌شده جدید خروجی داده شده و در ادامه نیز تابع *verification* صدا زده می‌شود تا درستی تابع اثبات شود و همان‌طور که مشاهده می‌کنید برنامه کاملا به درستی اجرا شده است.